Simultaneously Detecting Node and Edge Level Anomalies on Heterogeneous 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*—In complex systems like social media and financial transactions, diverse entities (users, groups, products) interact through a multitude of relationships (friendships, comments, purchases). These interactions can be represented by heterogeneous graphs (graphs with many node and edge types). In many real-world applications, these graphs may contain unusual patterns or anomalies. Detecting anomalies, both entity (node) level and interaction (edge) level anomalies, in these graphs are important, as their occurrence may have serious implications. Node-level anomalies may indicate abnormal behavior from a specific entity, such as unexpected activity that could suggest fraud. Edge-level anomalies may signify unusual interactions, like unexpected changes in interaction frequency or pattern, potentially indicating collaborative fraud like collusion.**

Unfortunately, existing graph neural network anomaly detection models focus only on homogeneous graphs and consider only node-level detection, rendering them incapable of harnessing the full complexity of heterogeneous graph data. To address this limitation, we present a new graph neural network model that capable of simultaneously detecting node-level and edge-level anomalies on heterogeneous graphs, by harnessing the rich information in the entities and relations. We develop our model as a type of graph autoencoder with a customized architecture design to enable the detection of node-level and edge-level anomalies simultaneously. Our graph neural network structure is scalable, facilitating its application in large real-world scenarios. Finally, our method outperforms previous anomaly detection methods in the experiments.

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

I. Introduction

Anomaly detection plays a crucial role in a variety of real-world applications, including fraud detection, network intrusion detection, e-commerce platform abuse detection, and other applications across diverse domains such as manufacturing, healthcare, insurance, and medicine [1]–[4]. In many of these applications, the datasets are often represented as heterogeneous graphs, where nodes and edges depict a variety of entities and their relationships, respectively. For instance, in a social media network, nodes might represent users, groups, and pages, while edges could signify different types of interactions such as likes, comments, shares, or friendships. Similarly, on an e-commerce platform, nodes could symbolize customers, products, and sellers, and edges could indicate purchase history, product viewing, or interactions between customers and sellers. In a cybersecurity network, nodes could represent different devices, servers, and users, and edges could denote communication or data transfer between them.

The heterogeneous graphs created by the interaction in the real-world scenarios above are usually rich in information. This information can be represented as node and edge features, providing a comprehensive view of the applications. For example, in the context of an e-commerce platform, node features could include customer demographics, product categories, or seller ratings. Edge features, on the other hand, could represent the frequency of purchases, the number of product views, or the nature of customer-seller interactions, such as communication frequency or review scores.

Across various domains, detecting anomaly patterns on these heterogeneous graphs is important, as the occurrence of these anomaly behaviors may have serious implications. For example, the occurrence of anomaly behaviors in an e-commerce platform may signal financial frauds such as stolen credit cards or money laundering [2], while in a network security system, anomalous events may indicate security breaches [3]. Additionally, it is also crucial to have capability in detecting anomaly patterns at both node-level and edge-level. Anomalies at the node-level could suggest abnormal behavior from a specific entity. For example, an entity might start behaving differently or exhibit unexpected activity that could indicate fraudulent behavior. Similarly, anomalies at the edge-level could indicate unusual interactions or relationships. For instance, the frequency or pattern of interactions between entities might change unexpectedly, suggesting collaborative fraudulent patterns, such as collusion. Therefore, having a high-performing anomaly detection system capable of simultaneously performing node-level and edge-level detections is beneficial in these domains. Unfortunately, current anomaly detection models are not capable of performing the task.

In most settings, anomaly detection is conducted without label supervision, i.e., in an unsupervised learning manner [4]. This approach provides advantages over supervised learning techniques due to its flexibility. In many applications, obtaining anomaly labels is challenging, as anomalous events occur infrequently [4]. Moreover, in real-world applications such as fraud detection, fraudsters are motivated to continually innovate their fraudulent methods. This makes supervised models, which rely on historical labels, being unable to detect these new innovations from the fraudsters [5].

In recent years, many graph neural network (GNN) models have been proposed for unsupervised anomaly detection tasks. However, the vast majority of the previous GNN models exclusively detect node-level anomalies on homogeneous graphs with node attributes only. This applies to reconstruction-based models (such as DOMINANT [6], AnomalyDAE [7], AEGIS [8], etc. [9]–[12]), contrastive-based models (like CoLA [13], CONAD [14], ANEMONE [15], etc. [16]–[20]), and others [21]–[24]. While a few GNN models have been proposed for scenarios beyond homogeneous graphs, they do not meet the requirements of the aforementioned applications. GraphBEAN [25] attempts to include multiple entities in detection but is limited to two entities (bipartite graphs). AHEAD [26] performs anomaly detection in heterogeneous graphs, but it is confined to node-level detection and cannot leverage edge features in its detection process.

Motivated by the shortcomings above, we propose a new graph neural network model that simultaneously performs node-level and edge-level anomaly detection on heterogeneous graphs. Our method is also capable of incorporating both node features and edge features into the detection. We name our method the Heterogeneous Node-and-Edge-Attributed Graph Neural Networks **(HeagNet)**. To the best of our knowledge, HeagNet is the first GNN-based anomaly detection model that can detect node-level and edge-level anomaly simultaneously on heterogeneous graphs.

To achieve the desired goal, we develop HeagNet as a type of autoencoder with a customize bottleneck to facilitate node-level and edge-level detection. The network is composed of one encoder and two decoders (feature and structure decoder). Each convolution layer in the encoder processes the heterogeneous graph with all its node and edge representations to produce new representations for each node and edge. However, the last layer of the encoder only produces latent node representations. These latent node representations are then used by the feature decoder to reconstruct the full graph with complete node and edge features. The latent representation is also used by the structure decoder to learn the graph's structure. This bottleneck construction ensures that the network learns the underlying patterns in the input graph and reconstructs the original graph based on these patterns, rather than simply copying the original input graph. The reconstructed errors of each edge and node in the graph serve as the basis for anomaly score creation.

We develop two versions of our model, HeagNet-C and HeagNet-A, which differ in how the model aggregates information from different edge types. HeagNet-A incorporates attention mechanisms in the aggregation, while HeagNet-C utilizes standard convolutional-based aggregation. Moreover, we design our model with scalability in mind, using customized negative edge sampling and neighboring subgraph sampling. This enables our models to be applied to large industrial graph data. Finally, we demonstrate the empirical benefits of our method over previous graph anomaly models on several publicly available heterogeneous graph datasets.

II. Related Works

A. Anomaly detection on homogeneous graphs

In recent years, numerous graph neural network (GNN) models have been proposed for unsupervised anomaly detection in homogeneous graphs. One of the pioneering models, DOMINANT [6], adopts an autoencoder-style architecture for graph anomaly detection, where the anomaly score is derived from the autoencoder's reconstruction error. AnomalyDAE [7] extends the architecture by employing two distinct autoencoders, one dedicated to the graph structure and the other to attributes. Many other GNN models follow these reconstruction-based models, including AEGIS [8], GAAN [9], GUIDE [10], GAD-NR [11], and ADA-GAD [12].

Another family of models uses contrastive-based losses as a means to train the anomaly model. The models work by contrasting positive pairs of related nodes or subgraphs, with negative pairs (unrelated nodes/subgraphs). The anomaly score for a node is calculated from the difference of the predicted score of its positive pair (should be close to 1 for normal node) and the scores for its negative pairs. This technique is used in many models such as: CoLA [13], CONAD [14], ANEMONE [15], GCCAD [16], GRADATE [17], and others [18]–[20]. Some other GNN models combine both reconstruction and contrastive loss (such as SL-GAD [23] and Mul-GAD [24]), while others use different approaches in anomaly detection (such as OCGNN [22] and AdONE [21]).

B. GNN models for heterogeneous graphs

Many GNN architectures have been proposed for heterogeneous graphs, particularly in supervised and semi-supervised settings. Metapath-based methods like HAN [27], HetGNN [28], GTN [29], MAGNN [30], and SeHGNN [31] capture the structural information via predefined composite relations (metapaths). Metapath-free methods like RGCN [32], RSHN [33], HetSANN [34], HGT [35], SimpleHGN [36], BA-GNN [37], and HINormer [38] added extra modules (e.g. attentions) to the standard GNN models for capturing semantic information like node type and edge type.

C. Anomaly detection on heterogeneous graphs

Even though many heterogeneous graphs models have been proposed in supervised and semi-supervised settings, only a few methods are designed for unsupervised anomaly detection in heterogeneous graphs. GraphBEAN [25] attempts to incorporate multiple entities into the detection but is limited to two entities in bipartite graphs. AHEAD [26] utilizes an HGT backbone for anomaly detection in heterogeneous graphs. However, it is limited to node-level detection only and cannot leverage edge features in the detection process.

III. Problem Formulation

We begin our problem formulation by describing our notations, starting with the definition of the node-and-edge-attributed heterogeneous graph. We are given a heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathcal{R})$, where $\mathcal{V}$ and $\mathcal{E}$ collect the nodes and edges, respectively. Each node $v_i \in \mathcal{V}$ is associated with a type

$t \in \mathcal{T}$, via type mapping functions $\delta : \mathcal{V} \rightarrow \mathcal{T}$. Similarly, each edge is also associated with an edge/relation type $r \in \mathcal{R}$, thus, we denote it using a triplet $e_{i,j}^{(r)} \triangleq (v_i, r, v_j) \in \mathcal{E}$. In terms of the attributes, each node $v_i \in \mathcal{V}$ and edge $e_{i,j}^{(r)} \in \mathcal{E}$ are also associated with a feature vector $\mathbf{x}_i^v$ and $\mathbf{x}_{i,j}^{e^{(r)}}$, respectively. Feature vectors for the same node/edge type have the same dimension, but they could vary across different node/edge types.

Next, we describe our neighboring operators. The node neighborhood operator, $\mathcal{N}^{(r)}(v_i)$, returns the list of all neighboring nodes from node v_i under relation (edge type) r, i.e., $\mathcal{N}^{(r)}(v_i) = \{v_j \in \mathcal{V} \mid (v_i, r, v_j) \in \mathcal{E} \vee (v_j, r, v_i) \in \mathcal{E}\}$. We also define edge neighborhood operator, $\mathcal{M}^{(r)}(v_i)$, that behaves similarly to $\mathcal{N}^{(r)}(v_i)$, but returns the edge connecting to the neighboring node, instead of the node itself, i.e.: $\mathcal{M}^{(r)}(v_i) = \{e_{i,j}^{(r)} \mid (v_i, r, v_j) \in \mathcal{E}; \forall v_j \in \mathcal{V}\} \cup \{e_{j,i}^{(r)} \mid (v_j, r, v_i) \in \mathcal{E}; \forall v_j \in \mathcal{V}\}$. Finally, we have edge type neighborhood function, $\mathcal{R}(v_i)$, that returns all relations (edge types) that are attached to node v_i, i.e. $\mathcal{R}(v_i) = \{r \in \mathcal{R} \mid (v_i, r, v_j) \in \mathcal{E} \vee (v_j, r, v_i) \in \mathcal{E}; \forall v_j \in \mathcal{V}\}$.

In our graph anomaly detection, we formulate the problem as a ranking problem with scoring functions, where a larger score means a higher degree of abnormality. Particularly, we need to produce scoring functions for both node-level and edge-level anomaly detections. As there are many node types and edge types in heterogeneous graphs, the scoring functions could be different for each node/edge type. Therefore, we have up to $|\mathcal{T}|+|\mathcal{R}|$ scoring functions for node-level and edge-level graph anomaly detections.

IV. Methodology

We will now describe our approach to addressing the aforementioned graph anomaly detection problem.

A. Architecture Overview

The main consideration in our model design is that the model should be capable of producing both node-level and edge-level scores simultaneously. Additionally, it must utilize all the rich information from the node and edge features of each node/edge type, as well as the graph structure. We design HeagNet as a reconstruction-based graph anomaly detection model. HeagNet architecture follows graph autoencoder architecture with a single graph encoder and two decoders: a feature decoder and a structure decoder. The key difference of HeagNet compared to other models lies in the type of graph it can take as input and produce as output (heterogeneous graph with node and edge attributes), and more importantly, the design of the autoencoder architecture to be able to achieve the desired goal of simultaneously performing node-level and edge-level anomaly detection on the graph. We will describe the details of our architecture choice in the following subsections.

B. Graph Convolution

The graph convolution operation in HeagNet is designed to incorporate signals from neighboring nodes and edges with various node/edge types, as well as their rich information in the form of node/edge features. This convolution operation also needs to produce a new representation for every node in each node type and every edge in each edge type.

Let $\mathbf{h}$ denote the intermediate representation vector where we use a superscript to indicate node/edge representation. Specifically, we use $\mathbf{h}_i^v$ to denote the representation of the node v_i, and $\mathbf{h}_{i,j}^{e^{(r)}}$ for the representation of the edge $e_{i,j}^{(r)}$. Additionally, we add another superscript to indicate the information about the layer. Specifically, for the k-th layer, the node and edge representation symbols are $\{\mathbf{h}_i^v\}^{(k)}$ and $\{\mathbf{h}_{i,j}^{e^{(r)}}\}^{(k)}$, respectively.

1) Intra-relation message passing: For the k-th layer, the graph convolution layer takes the previous layer representations $\{\mathbf{h}_i^v\}^{(k-1)}, \forall v_i \in \mathcal{V}$ and $\{\mathbf{h}_{i,j}^{e^{(r)}}\}^{(k-1)}, \forall e_{i,j}^{(r)} \in \mathcal{E}$ to compute the new node and edge representations. As there are many node/edge types, to create new node representation, we first perform intra-relation (edge type) message passing and aggregation. Let $\text{Msg}[\rightarrow v_i]^{(r,k)}$ denote the message that comes to node v_i via relation r in the k-th layer. This message comes from the neighboring nodes as well as the neighboring edges under relation r. We perform aggregation to the node and edge messages independently, and then concatenate them. We then pass the concatenated messages to a linear operation parameterized by $\mathbf{W}_v^{(r,k)}$ and $\mathbf{b}_v^{(r,k)}$. Finally, we normalize the output of the linear operation using batch normalization (BN) and pass it to a ReLU activation function as follows:

$$\begin{aligned}\text{Msg}[\rightarrow v_i]^{(r,k)} = \text{ReLU}\Big(\text{BN}\Big(\mathbf{b}_v^{(r,k)} + \\ \mathbf{W}_v^{(r,k)} * \Big\{\text{Agg}\big\{\{\mathbf{h}_j^v\}^{(k-1)} \mid \forall v_j \in \mathcal{N}^{(r)}(v_i)\big\} \\ \cup \text{Agg}\big\{\{\mathbf{h}_{i,j}^{e^{(r)}}\}^{(k-1)} \mid \forall e_{i,j}^{(r)} \in \mathcal{M}^{(r)}(v_i)\big\}\Big\}\Big)\Big)\end{aligned} \tag{1}$$

Here, $\cup$ represents concatenation operator, whereas Agg denote the aggregation function. This could be a simple aggregation function like Mean and Max, a combination of both, or more complex aggregation functions.

2) Inter-relation message passing: After computing intra-relation messages from different relations, we then perform inter-relation message passing to create new node representation. We aggregate the intra-relation messages for each relation connected to the target node v_i. We then concatenate it with the previous representation v_i followed by applying linear operation parameterized by $\mathbf{W}_v^{(k)}$ and $\mathbf{b}_v^{(k)}$. Similarly, we also apply batch normalization (BN) and ReLU activation to produce the new representation for v_i, as follow:

$$\begin{aligned}\{\mathbf{h}_i^v\}^{(k)} = \text{ReLU}\Big(\text{BN}\Big(\mathbf{b}_v^{(k)} + \mathbf{W}_v^{(k)} * \Big\{\{\mathbf{h}_i^v\}^{(k-1)} \\ \cup \text{Agg}\big\{\text{Msg}[\rightarrow v_i]^{(r,k)} \mid \forall r \in \mathcal{R}(v_i)\big\}\Big\}\Big)\Big)\end{aligned} \tag{2}$$

In addition to the inter-relation message passing and aggregation above, we also create another message passing scheme based on an attention mechanism. This attention mechanism tries to learn the importance of different relations to the target node, by putting different weights for each relation in the aggregation process. Specifically, we compute the attention of the target node v_i to each relation r in the k-th layer, $\alpha_i^{(r,k)}$,

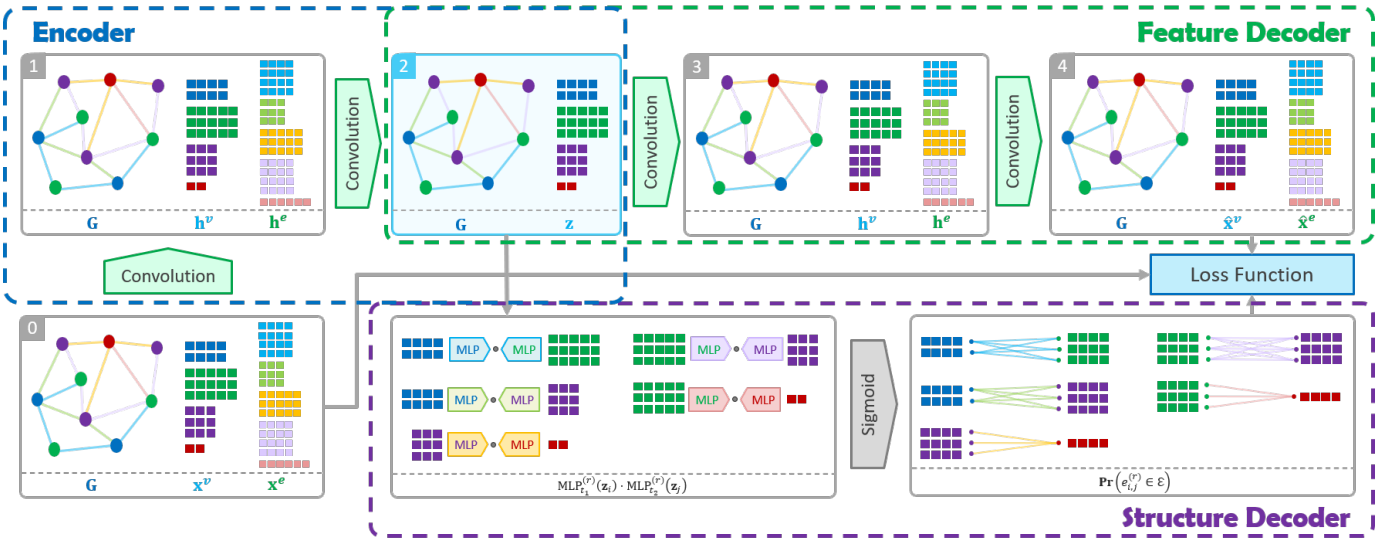


Fig. 1: HeagNet architecture with 4 graph convolution layers.

as a softmax over the intra-relation messages, $\text{Msg}[\rightarrow v_i]^{(r,k)}$, multiplied by a learnable weight vector $\Phi^{(r,k)}$, i.e.:

$$\alpha_i^{(r,k)} = \frac{\exp\left\{\text{Msg}[\rightarrow v_i]^{(r,k)} \cdot \Phi^{(r,k)}\right\}}{\sum_{s\in\mathcal{R}(v_i)} \exp\left\{\text{Msg}[\rightarrow v_i]^{(s,k)} \cdot \Phi^{(s,k)}\right\}} \quad (3)$$

We then use the attention variables to perform weighted aggregation of the messages that come from each relation. To produce the new representation for node v_i, we add the aggregated message with the node's previous representation multiplied by a weight matrix $\mathbf{W}_l^{(k)}$, and then apply normalization and activation function, i.e.:

$$\{\mathbf{h}_i^v\}^{(k)} = \text{ReLU}\Big(\text{BN}\Big(\sum_{r\in\mathcal{R}(v_i)} \alpha_i^{(r,k)} \cdot \text{Msg}[\rightarrow v_i]^{(r,k)} + \mathbf{W}_l^{(k)} * \{\mathbf{h}_i^v\}^{(k-1)}\Big)\Big) \quad (4)$$

We name the version of our model that uses the attention mechanism above as HeagNet-A, whereas the one that uses the regular aggregation (Eq. (2)) as HeagNet-C.

3) Computing the next edge representation: We have described the method for computing the next node representation. In this subsection, we will describe the technique to compute the next edge representation. The messages that go to edge $e_{i,j}^{(r)}$ simply comes from the nodes connected to the edge, i.e., v_i and v_j, and its own previous representation. For computing the next edge representation, we simply concatenate the messages and then apply a linear operation parameterized by $\mathbf{W}_e^{(r,k)}$ and $\mathbf{b}_e^{(r,k)}$. Finally, we apply a normalization and activation function as follows:

$$\text{Msg}[\rightarrow e_{i,j}^{(r)}]^{(k)} = \left\{\{\mathbf{h}_i^v\}^{(k-1)} \cup \{\mathbf{h}_j^v\}^{(k-1)} \cup \{\mathbf{h}_{i,j}^{e^{(r)}}\}^{(k-1)}\right\} \quad (5)$$

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

C. Network Architecture

We have described HeagNet's convolution operations and message passing scheme, such that it can take heterogeneous graphs with node and edge attributes as input and produce similar graph structures as the output. In this section, we will describe how we assemble the basic convolution operation above into a full network architecture capable of performing node-level and edge-level anomaly detection.

HeagNet is designed as a graph autoencoder with one encoder and two decoders (feature and structure decoder). It consists of K (even number) convolution layers, where we use half of them for the encoder and the other half for the feature decoder. In each convolution layer, the layer takes the heterogeneous graph with all its node and edge representations. It then performs a convolution operation to produce new representation for each node and each edge. The exception is on the last layer of the encoder. Instead of producing both node and edge representations, it only produces latent representations for the nodes, without generating edge representations. Additionally, each node in the last encoder layer is not allowed to use its own previous representation.

The node-only latent representations constructed by the encoder are then picked up by the feature decoder to produce the full reconstructed graph with complete node features and edge features via a series of graph convolutions. Additionally, we use the structure decoder to allow the model to learn the structure of the graph, since in the feature decoder, the graph structure is given as the basis to perform convolution operations.

This construction serves as the bottleneck of the HeagNet autoencoder architecture, so that the network cannot just copy the original input graph to do the reconstruction. It has to learn the underlying patterns in the input graph and reconstruct the original graph based on the discovered patterns. Additionally, by the construction above, the node latent representations constructed by the encoder are forced to learn its own node feature, all K-hop neighboring nodes and their node features, as well as the neighboring edges and their edge features, and the local graph structures surrounding the node.

1) Encoder: The encoder consists of $K/2$ graph convolution layers. We set the input to the first layer as the original graph's node and edge features, i.e., $\{\mathbf{h}_i^v\}^{(0)} = \mathbf{x}_i^v$ and $\{\mathbf{h}_{i,j}^{e^{(r)}}\}^{(0)} = \mathbf{x}_{i,j}^{e^{(r)}}$. The convolution operations for the next layers follow our previous description in Section IV-B. The exception is on the last layer, where we only produce node representation and we do not allow the node to copy its previous representation. Specifically, we replace Eq. (2) with:

$$\{\mathbf{h}_i^v\}^{(K/2)} = \text{ReLU}\Big(\text{BN}\Big(\mathbf{b}_v^{(K/2)} + \mathbf{W}_v^{(K/2)} * \Big\{ \cup \text{Agg}\big\{\text{Msg}[\rightarrow v_i]^{(r,K/2)} \mid \forall r \in \mathcal{R}(v_i)\big\}\Big\}\Big)\Big) \tag{7}$$

for HeagNet-C, or we replace Eq. (4) with:

$$\{\mathbf{h}_i^v\}^{(K/2)} = \text{ReLU}\Big(\text{BN}\Big(\sum\nolimits_{r\in\mathcal{R}(v_i)} \alpha_i^{(r,K/2)} \cdot \text{Msg}[\rightarrow v_i]^{(r,K/2)}\Big)\Big) \tag{8}$$

for HeagNet-A. We then set the latent representation $\mathbf{z}$ as the output of the layer, i.e. $\mathbf{z}_i = \{\mathbf{h}_i^v\}^{(K/2)}, \forall v_i \in \mathcal{V}$.

2) Feature Decoder: The feature decoder also consists of $K/2$ convolution layers. The first layer takes the latent node representations from the encoder without accepting edge representations, as the encoder only produces node representations. Specifically, we replace Eq. (1) and Eq. (5) with:

$$\text{Msg}[\rightarrow v_i]^{(r,\frac{K}{2}+1)} = \text{ReLU}\Big(\text{BN}\Big(\mathbf{b}_v^{(r,\frac{K}{2}+1)} + \mathbf{W}_v^{(r,\frac{K}{2}+1)} * \Big\{\text{Agg}\big\{\mathbf{z}_j \mid \forall v_j \in \mathcal{N}^{(r)}(v_i)\big\}\Big\} \tag{9}$$

$$\text{Msg}[\rightarrow e_{i,j}^{(r)}]^{(\frac{K}{2}+1)} = \{\mathbf{z}_i \cup \mathbf{z}_j\}. \tag{10}$$

The remaining layers follow the regular convolution operations described in Section IV-B. The output of the last layer becomes the reconstructed node and edge features, i.e. $\hat{\mathbf{x}}_i^v = \{\mathbf{h}_i^v\}^{(K)}$ and $\hat{\mathbf{x}}_{i,j}^{e^{(r)}} = \{\mathbf{h}_{i,j}^e\}^{(r,K)}$.

3) Structure Decoder: The structure decoder is tasked to enforce the latent node representation $\mathbf{z}$ to learn the structure of the graph. This is needed as the feature decoder takes the structure of the graph as given. The structure decoder is required to predict if an edge exists between two nodes, based on the latent representation of each node. Specifically, for each relation r, we create two multi-layer perceptrons (MLPs), one for the left side node type t_1 of the relation ($\text{MLP}_{t_1}^{(r)}$), another one from the right side node type t_2 ($\text{MLP}_{t_2}^{(r)}$). For each node in the left node type $\{v_i \in \mathcal{V} \mid \delta(v_i) = t_1\}$ and each node in the right node type $\{v_j \in \mathcal{V} \mid \delta(v_j) = t_2\}$, the structure decoder defines the probability of the existence of the edge connecting v_i and v_j under relation r as:

$$\Pr\left(e_{i,j}^{(r)} \in \mathcal{E}\right) = \text{sigm}(\text{MLP}_{t_1}^{(r)}(\mathbf{z}_i) \cdot \text{MLP}_{t_2}^{(r)}(\mathbf{z}_j)) \tag{11}$$

In practice, most node pairs are not connected via an edge. Therefore, we do not need to consider all pairs to learn the graph structure. Instead, we use all pairs of connected nodes in the graph as the "positive" instances, and just sample a set of non-connected nodes as the "negative" instances. Let us define the set of non-connected node pairs as $\mathcal{E}^- = \left\{e_{i,j}^{(r)} \notin \mathcal{E} \mid \forall v_i, v_j \in \mathcal{V}; \forall r \in \mathcal{R}\right\}$. We then define a set of instances, $\widetilde{\mathcal{E}}^{\pm} = \mathcal{E} \cup \text{Sample}(\mathcal{E}^-)$, that the structure decoder needs to operate on. Specifically, we use a uniform random sampling to get the instances in $\mathcal{E}^-$ with a pre-specified number of instances, such as a small multiple of the number of edges in the graph ($\mathcal{E}$).

D. Model Training

We have described our network architecture. In this section, we will explain the procedure to train the network.

1) Loss Function: To train HeagNet, we use a reconstruction error loss function that compares the output of the reconstructed node and edge features from the feature decoder, $\hat{\mathbf{x}}_i^v$ and $\hat{\mathbf{x}}_{i,j}^{e^{(r)}}$, with the original graph features using mean squared error (MSE). The loss also compares the probability of the existence of an edge produced by the structure decoder, $\Pr(e_{i,j}^{(r)} \in \mathcal{E})$, with the ground truth label of the existence of the edges in the original graph, via binary cross entropy (BCE) loss. Specifically, our loss is defined as follow:

$$\begin{aligned}\mathcal{L} =& \frac{1}{|\mathcal{V}|}\sum_{v_i\in\mathcal{V}} \tfrac{1}{d_t}\|\mathbf{x}_i^v - \hat{\mathbf{x}}_i^v\|_2^2 + \frac{1}{|\mathcal{E}|}\sum_{e_{i,j}^{(r)}\in\mathcal{E}} \tfrac{1}{d_r}\|\mathbf{x}_{i,j}^{e^{(r)}} - \hat{\mathbf{x}}_{i,j}^{e^{(r)}}\|_2^2 \\ &+ \frac{\eta}{|\widetilde{\mathcal{E}}^{\pm}|}\sum_{(i,j)\in\widetilde{\mathcal{E}}^{\pm}} \Big\{ -\mathbb{I}\left(e_{i,j}^{(r)} \in \mathcal{E}\right)\log\left(\Pr(e_{i,j}^{(r)} \in \mathcal{E})\right) \\ &- \mathbb{I}\left(e_{i,j}^{(r)} \notin \mathcal{E}\right)\log\left(1 - \Pr(e_{i,j}^{(r)} \in \mathcal{E})\right)\Big\},\end{aligned} \tag{12}$$

where $\mathbb{I}$ indicates the indicator function, whereas d_t and d_r denote the dimension of the features for the particular node type and edge type, respectively. The constant η is used for balancing the feature decoder's MSE loss and the structure decoder's BCE loss.

2) Forward and Backward Propagation: We have described all the components of HeagNet. In the training procedure, we perform forward propagation by feeding a node-and-edge-attributed heterogeneous graph to the model. The forward propagation process is illustrated in Figure 1, using a 4 layer HeagNet as an example. The encoder takes the input graph $\mathcal{G}$, with its node and edge features ($\mathbf{x}^v$ and $\mathbf{x}^e$, respectively) for each node type and edge type. The different colors of the node and edges in the graph represent different node/edge types. The matrix illustration of the node/edge features also reflect different colors representing different node/edge types. The dimension of node/edge features in each node/edge type can also be different, reflected in the matrix illustration as well.

The first encoder layer then processes the input graph and constructs new node and edge representations ($\mathbf{h}^v$ and $\mathbf{h}^e$ respectively) for each node/edge type, which then be passed to the next layer. The second (last) encoder layer takes these representations as its input (in addition to the graph structure) to produce the node latent representation $\mathbf{z}$ for each node type. Note that the layer does not produce edge representations. The feature decoder then takes $\mathbf{z}$ for each node type and process it to a series of convolution layers, to produce the reconstructed node and edge features ($\hat{\mathbf{x}}^v$ and $\hat{\mathbf{x}}^e$, respectively) for each node/edge type.

TABLE I: Dataset properties.

Dataset	#(node, edge) type	#node	#edge	avg deg.	avg (node, edge) dim.	node +ratio	edge +ratio
Telecom-Small	(4, 3)	80,380	890,000	11.1	(370, 50)	0.012	0.005
Reddit	(4, 4)	64,180	76,193	1.2	(384, 384)	0.010	0.011
Brightkite	(5, 4)	125,467	608,466	4.8	(10, 8)	0.026	0.047
Gowalla	(5, 4)	282,812	2,092,019	7.4	(10, 8)	0.018	0.012
Telecom-Large	(4, 3)	170,380	8,900,000	52.2	(370, 50)	0.017	0.002

Another path from the latent representation goes to the feature decoder, where we have a pair of MLPs for each edge type (relation). For each relation, a pair of MLPs takes the node latent representation of the node types connected by the relation as the input. It then performs a dot product on the outputs of the MLPs for each node in the left-side node type, with the ones from the right-side node type of the relation, followed by a sigmoid function, to predict the probability of the existence of an edge connecting the nodes. These probability outputs alongside the reconstructed node and edge features for each node/edge type are then passed to a loss function to produce the objective values. This objective is then used to train the model. For computing backward propagation (gradient) and performing updates to the model, we resort to a deep learning framework.

E. Anomaly Score

After we finish the training, we produce anomaly scoring functions for every node type and every edge type in the graph. The scoring functions are computed by first performing a forward propagation of the network with the input graph, and then computing the individual node/edge reconstruction error. Similar to other autoencoder-based models, as the model optimizes an average reconstruction loss, it drives down the loss of the common patterns found in the graph while being relatively tolerant to cases that occur infrequently. Therefore, at inference, normal patterns that commonly occur in the graph can be easily reconstructed, i.e., having low reconstruction error. On the other hand, anomalous patterns, which rarely occur, suffer from high reconstruction errors.

We first define the edge scoring functions as the weighted combination of the edge features' MSE and the BCE of predicting if an edge should exist in the graph. We then define the scoring function for the nodes as the combination of the reconstruction error of its node feature and aggregation of the anomaly score of all edges connected to the node under all relation types, i.e.:

$$\begin{aligned}\text{score}_e^{(r)}(e_{i,j}^{(r)}) &= \tfrac{1}{d_r}\|\mathbf{x}_{i,j}^{e^{(r)}} - \hat{\mathbf{x}}_{i,j}^{e^{(r)}}\|_2^2 - \eta \log(\Pr(e_{i,j}^{(r)} \in \mathcal{E})) \\ \text{score}_v^{(t)}(v_i) &= \tfrac{1}{d_t}\|\mathbf{x}_i^v - \hat{\mathbf{x}}_i^v\|_2^2 + \underset{\substack{r\in\mathcal{R}(v_i)\\ e_{i,j}^{(r)}\in\mathcal{M}^{(r)}(v_i)}}{\text{Agg}} \text{score}_e^{(r)}(e_{i,j}^{(r)}).\end{aligned} \tag{13}$$

The aggregation function AGG could be MEAN or MAX, depending on the need of the applications.

F. Scalability via Neighborhood Sampling

We design HeagNet with scalability in mind. The first component of our scalability design is in the structure decoder, where we do not need to store the full adjacency matrices for representing the edges, unlike many previous models [6], [7], [26], [39]. The size of adjacency matrices could be huge for large industrial applications, particularly if they are stored as dense matrices. The second component of our scalability design is that we use neighborhood sampling during the training process. Specifically, in each forward propagation, we sample a subgraph from the full heterogeneous graph and update the weights based on this subgraph. To create the subgraph, we first sample a set of nodes from the pre-specified node types, and then expand the subgraph by iteratively sampling the neighborhood of the subgraph multiple times. In addition, we also design layer-wise and relation-wise sampling for the inference process, as we need to compute anomaly scores for every single node and edge in every single node/edge type in the graph.

V. EXPERIMENTS

We evaluate HeagNet to perform anomaly detection on several publicly available datasets. We evaluate the models on both node-level and edge-level anomaly detection. The input graphs to the models are heterogeneous graphs with both node features and edge features.

A. Experiment Setup

1) Datasets: In our experiments, we use several publicly available datasets. The Telecom dataset [40] describes the relationship between users and behaviors in a telecommunication network, containing four types of nodes: user, package, app, cell, and three types of edges: user-buy-package, user-use-app, user-live-cell. The Reddit dataset [41] contains user interactions on the Reddit forum. The users and subreddits are grouped into multiple clusters. Each cluster becomes a node type. The edge types reflect the interaction between user nodes and subreddits nodes. The edge features are created by processing the text in interaction using the SentenceBERT language model [42] to produce sentence embeddings. The node features are created by aggregating edge features connected to the node.

The Brightkite and Gowalla datasets [43] describe user interactions on location-based social networks, Brightkite and Gowalla. From the raw latitude and longitude location data, we construct location nodes using GeoHash representation (level 6) of the coordinates [44], [45]. We cluster on the coordinates to group the geohash nodes into 4 groups. The edge features are created from the activity statistics of a user in a geohash location. The features for the user node contain the activity statistics of a user on the platform. Similarly, the features for

the geohash node contain the statistics of check-in activities that occur inside the geohash.

Detailed information on the dataset properties, such as the number of node/edge types, the number of nodes/edges, the average node degree, and the average dimension of the node/edge features, is available in Table I. Due to the size of the Telecom dataset, we create two different versions of the dataset: Telecom-Large containing the full-size dataset, and Telecom-Small containing the sampled version. All of the datasets contain 4-5 node types and 3-4 edge types. Every node type and edge type have their own node/edge features. In all of the datasets, we perform both node-level and edge-level anomaly detection.

2) Baselines: As we are pioneering the task of simultaneous node and edge level anomaly detection on heterogeneous graphs, there are no baseline available that are specifically designed for the task. Therefore, for the purpose of our experiment, we select baseline methods that do not necessarily have the capability of performing the task, then adjust and modify the methods or the datasets to simulate the necessary requirement of node and edge level detection on heterogeneous graph datasets. We believe that our experiments can still provide some insight on the benefit of our proposed model, as in real-world problems, practitioners may also face similar challenges of either modifying available models to fit the problem space, or modifying the problem space to fit the available models.

AHEAD [26] is the closest method from our task, as it also works on heterogeneous graphs. However, it only performs node-level detection and can only accept node features. For AHEAD, we use the node features in the heterogeneous graphs and allow the model to produce node anomaly scores. The anomaly scores for each edge are produced by taking the average score of the two nodes connected to it. Isolation Forest [46] is a classical tree-based model that is popular in many industrial applications. As the model does not use graph structure, we independently create separate models for each node type and each edge type in the graph.

Additionally, we also select GNN models for node-level anomaly detection homogeneous graphs, as the majority of previous models work on this setup, despite the setup differs a lot from our task. To simulate our anomaly detection task on these homogeneous graph models, we convert the original heterogeneous graphs to homogeneous graphs by removing the node/edge type information. For the node attribute, we make the dimension of the features equal by first concatenating different features from different node types into a single big block diagonal matrix, and then running principal component analysis (PCA) to reduce the dimensionality of the matrix. Similar to the AHEAD case, we let the model produce node scores, and use the same technique for producing edge scores. For the method selection, we select some of the most popular methods in homogeneous graph anomaly detection, the DOMINANT [6], AnomalyDAE [7], and CONAD [14]. Therefore, we compare seven models in our experiment: the tree based Isolation Forest; the node-level heterogeneous graph model, AHEAD; three node-level homogeneous graph models; as well as two versions of our model, HeagNet-C and HeagNet-A.

3) Anomaly injection: As commonly done in many previous anomaly detection studies [6]–[20], we inject anomalies into the heterogeneous attributed graphs, due to no ground truth anomalies in the datasets. We specifically follow the anomaly injection techniques described in [6], [25], with some modifications for heterogeneous graph use cases. There are two main components of the anomalies that we inject into the graph: the topological structure and attribute anomalies. The topological anomaly injection is done relation-wise. For each relation (edge type), we randomly select a small number of nodes and add a full or partial dense block involving the selected nodes. Small dense structures in a graph are typical anomalous substructures in which the interactions are much more intense than average [6]. For injecting attribute anomalies, we use two different techniques: the outside of a confidence interval technique [47], [48], and the scaled Gaussian noise technique [49]. We repeatedly inject the small block anomalies with random selections of attribute anomalies 15-40 times, to simulate multiple anomaly occurrences in the dataset, each with its own properties. All the edges and nodes involved in the injected anomalies are labeled as anomalous. The ratio of positive (anomalous) cases compared to all cases is shown in the last two columns of Table I.

4) Implementation: We implement our method on top of PyTorch [50] and PyTorch Geometric [51] frameworks. For the AHEAD, we use the author's implementation [26], whereas, for the Isolation Forest, we use Scikit-Learn implementation. For the homogeneous GNN baselines (DOMINANT, AnomalyDAE, and CONAD), we use PyGOD [39] implementation of the algorithms. Most of the experiments are conducted in a single Linux machine from the AWS service with 16 vCPU cores, 60 GB of RAM and an NVIDIA Tesla V100 GPU. The exception is for the large datasets, where we use machines with larger RAM, up to 120 GB.

5) Experiment details: In our experiments, we conduct 10 runs for each dataset, except for the Telecom-Large dataset where we perform 5 runs due to its size. In each run, we repeat the anomaly injection procedure to ensure that each run has a different set of anomalies. We maintain the same set of anomalies across our method and all the baselines to ensure comparability of the results. During the experiment, we set the parameter that balances feature and structure decoder, $\eta = 0.2$, for our method. For AHEAD, DOMINANT, and AnomalyDAE, we set the similar parameters as suggested in their respective papers. To compute node anomaly scores in our model, we employ the Max aggregation.

We conduct the experiment using 4 layers HeagNet (both HeagNet-C and HeagNet-A), where we use 2 convolution layers for the encoder, and 2 convolution layers for the feature decoder. Each MLP used in the structure decoder also utilizes 2 dense layers. Similarly, the GNN-based baselines (DOMINANT, AnomalyDAE, CONAD, and AHEAD) also use 4 layers. To train HeagNet and all GNN baselines, we use Adam optimizer. The final learning rate is decided after running a few

TABLE II: The mean (and stdev.) of the Average-AUCPR metrics over multiple experiment runs in each dataset.

Model	IsoForest		DOMINANT		AnomalyDAE		CONAD		AHEAD		HeagNet-C		HeagNet-A	
Dataset	Node	Edge	Node	Edge	Node	Edge	Node	Edge	Node	Edge	Node	Edge	Node	Edge
Telecom-Small	0.924 (0.06)	0.556 (0.04)	0.428 (0.05)	0.196 (0.06)	0.132 (0.04)	0.036 (0.04)	0.427 (0.05)	0.196 (0.06)	0.942 (0.05)	0.597 (0.06)	**0.970** (0.02)	**0.715** (0.07)	0.965 (0.04)	0.711 (0.07)
Reddit	0.955 (0.05)	0.770 (0.17)	0.644 (0.15)	0.705 (0.08)	0.533 (0.10)	0.567 (0.07)	0.644 (0.15)	0.705 (0.08)	0.949 (0.05)	0.545 (0.09)	**0.968** (0.03)	0.788 (0.17)	0.963 (0.02)	**0.791** (0.15)
Brightkite	0.893 (0.07)	0.547 (0.15)	0.731 (0.12)	0.569 (0.05)	0.185 (0.04)	0.235 (0.09)	0.719 (0.12)	0.568 (0.05)	0.616 (0.11)	0.202 (0.09)	**0.928** (0.05)	**0.590** (0.12)	0.907 (0.05)	0.534 (0.10)
Gowalla	0.845 (0.05)	0.246 (0.06)	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	**0.952** (0.03)	**0.445** (0.09)	0.930 (0.03)	0.310 (0.07)
Telecom-Large	0.945 (0.07)	0.493 (0.10)	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	**0.964** (0.05)	**0.642** (0.11)	0.961 (0.05)	0.616 (0.10)

experiments using different learning rates (i.e., 0.0001, 0.001, 0.003, 0.01, 0.03). For HeagNet and all GNN models, the dimension of latent variables for every node is set to 64. The training procedures use a batch size of 1024. 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.). The random seed use in all experiments and all models is 0. Lastly, 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 (subgraph) batch, for the edge prediction objective.

B. Experiment Results

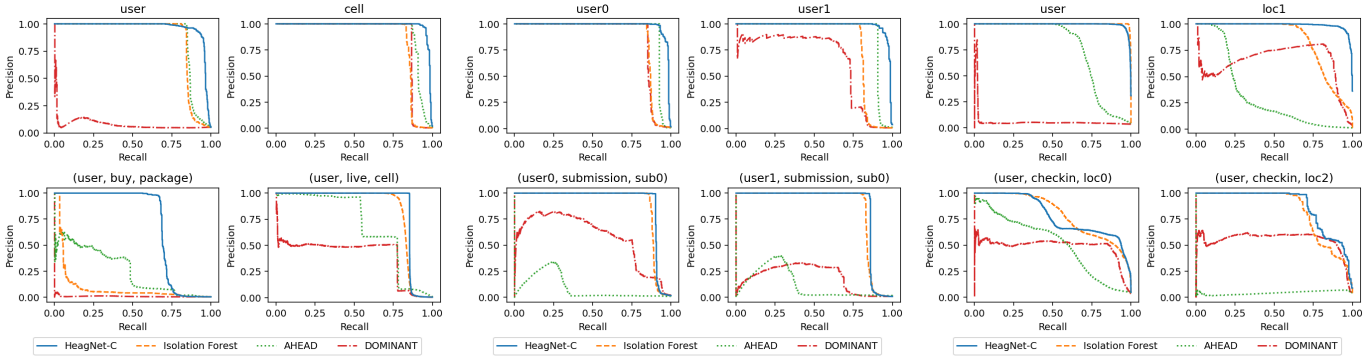
1) Evaluation metric: Since different methods have different scoring systems, we employ a ranking-based metric to evaluate the models. Specifically, we use the Area Under the Precision-Recall Curve (AUCPR) as the evaluation metric. Precision and recall provide better metrics compared to the alternatives as most anomaly detection applications are highly imbalanced, with the number of anomaly cases being far less than the number of normal cases. Moreover, practitioners often adjust the balance of precision and recall of a deployed model to achieve the best business impact. A threshold-free metric like AUCPR is suitable for anomaly detection evaluation in such cases. Since the heterogeneous graphs in our datasets contain many node and edge types, we report the Average-AUCPR metric over all node/edge types in the graphs.

2) Overall results: The mean and standard deviation of the Average-AUCPR over multiple runs for each dataset are presented in Table II. The results are divided into two sections: the node-level and edge-level Average-AUCPR. Alongside the baselines, we provide the results of the two versions of our model, HeagNet-C and HeagNet-A, which differ in their inter-relation message passing scheme as described in Section IV-B. HeagNet-C purely uses the convolution operation, whereas HeagNet-A incorporates an attention mechanism in its message passing scheme. The bold numbers in the table indicate the best results, while the underlined numbers indicate the second-best results.

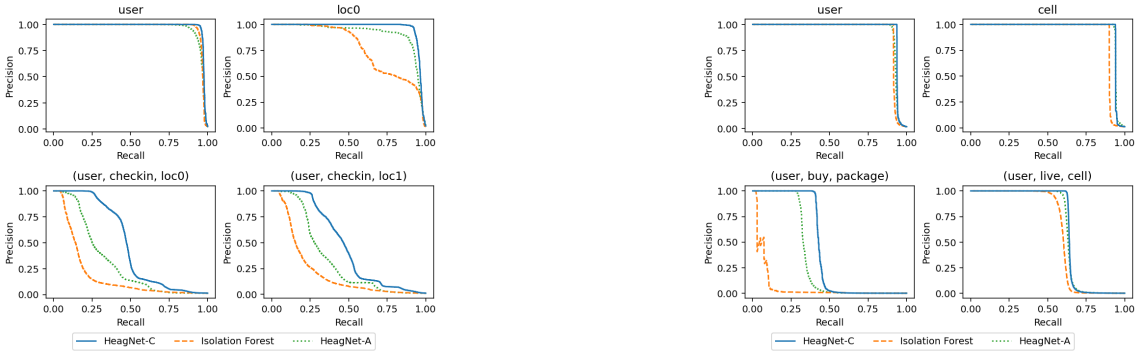
As we can see from the table, HeagNet-C performs the best in all datasets in node-level anomaly detection. HeagNet-C also performs the best in nearly all datasets in edge-level anomaly detection, except in the Reddit dataset where it achieves the second-best result, slightly trailing HeagNet-A. The attention version of our model, HeagNet-A, achieves the best or the second-best results in nearly all datasets in both node-level and edge-level anomaly detection, except in Brightkite's edge-level detection.

3) The effect of the attention mechanism: We observe that the attention mechanism we introduce in HeagNet-A generally does not improve the performance of our model compared with the standard graph convolution. This is despite many successful applications of attention models in heterogeneous graphs under supervised and semi-supervised learning settings [33]–[35], [52]. One reason could be the difference in the learning objectives between supervised learning models and unsupervised anomaly detection. In a supervised learning setting, the attention mechanism helps to classify a target node by increasing the influence of relevant information from the neighbors and decreasing the influence of non-relevant ones via the attention weights. In unsupervised anomaly detection, the goal is to describe a node by its interactions with the surrounding neighbors. If it cannot be easily described by its surroundings (i.e., having high reconstruction error), it is more likely to be anomalous. The attention mechanism may be counterproductive in this setting, as it may hide the anomalous signals from the surrounding neighbors as they may be considered as irrelevant in reducing the reconstruction error. This could make the anomaly score for the target node small, despite the existence of the anomalous signal.

4) Node-level and edge-level detection results: In edge-level anomaly detection, both HeagNet-C and HeagNet-A significantly outperform all baselines by a large margin in several datasets, particularly the Telecom-Small, Telecom-Large, and Gowalla. They also maintain a relatively significant lead over all GNN baselines in node-level anomaly detection. These results are expected, as our methods, HeagNet-C and HeagNet-A, are the only methods specifically designed for the task and capable to incorporate full information in the graphs (both node and edge attributes, graph structure, and node/edge type information). The other GNN baselines: AHEAD, DOMINANT, AnomalyDAE, and CONAD are not



(a) Telecom-Small (b) Reddit (c) Brightkite



(d) Gowalla (e) Telecom-Large

Fig. 2: Precision-recall curves of the node-level and edge-level anomaly detection on each dataset.

able to use the edge features in their detection. In addition, the homogeneous GNN models are not able to utilize the node/edge type information in their detection. Since Isolation Forest is a classical model, it cannot utilize the graph structure information. However, it can utilize both node features and edge features independently.

5) Comparison among the baselines: Among the GNN baselines, AHEAD performs relatively better compared to the homogeneous GNN models on node-level anomaly in the Telecom-Small, Reddit, and Brightkite datasets, and on edge-level anomaly in the Telecom-Small dataset. One reason could be the diversity of feature information among different node/edge types in the Telecom dataset, both in node and edge features. Therefore, the AHEAD model, which can utilize the node/edge type information, is able to perform better than the homogeneous GNN models. In the Reddit and Brightkite datasets, even though their node features are relatively diverse among different node types, their edge features are similar across different edge types. This may prevent AHEAD from capitalizing the edge type information to improve its performance. The attribute information is crucial in anomaly detection. As a result, the Isolation Forest method, which fully utilizes node and edge features, is able to achieve reasonably good performance across different datasets. However, as it is not able to utilize the graph structure and information from neighboring nodes/edges, its performance cannot surpass our models.

6) Precision-Recall Curve: In the previous results, we reported the Average-AUCPR metrics to perform method comparisons. The AUCPR metric provides a good overall look at the prediction performance. However, when it comes to the deployment of an anomaly detection system, practitioners may want to see the precision vs. recall trade-off at a given threshold. The precision-recall curve provides a tool to perform trade-off analysis at multiple thresholding points. We present the precision-recall curve on selected node and edge level anomaly detections (selected node/edge types) for a single experiment run in all datasets in Figures 2. In the plots, we selected baselines that perform competitively to be included as a line chart in the plot. As we can see from the figures, in most of the cases in all of the datasets, HeagNet provides a better precision-recall tradeoff compared to the baselines at most thresholding points. In some cases, like in Telecom-Small, Gowalla, and Telecom-Large, HeagNet provides significantly better results compared to the baselines. These results further confirm the benefits of HeagNet for node-level and edge-level anomaly detection on heterogeneous graphs.

7) Scalability: From a scalability perspective, our models, HeagNet-C and HeagNet-A, are capable of running on large industrial datasets, such as in the Gowalla and Telecom-Large datasets. The other GNN baselines, however, are not scalable to this size. We have tried to run them on larger machines without success due to out-of-memory (OOM) problems. One of the main reasons is that in their implementation, they require

storing the full adjacency matrices in a dense matrix format. This requirement is not feasible for large size graph data.

VI. Conclusions

We have proposed HeagNet, a graph neural network model capable of simultaneously performing node-level and edge-level anomaly detection in heterogeneous graphs with node and edge attributes. Our model is the first GNN model capable of performing the task in heterogeneous graphs. We demonstrated the performance benefits over the baselines in detecting anomalies in several public datasets. For future directions, we plan to extend our investigation for anomaly detection on heterogeneous temporal 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] K. Ding, J. Li, R. Bhanushali, and H. Liu, "Deep anomaly detection on attributed net." in *SDM*, 2019.

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

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

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

[10] 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.

[11] A. Roy, J. Shu, J. Li, C. Yang, O. Elshocht, J. Smeets, and P. Li, "Gad-nr: Graph anomaly detection via neighborhood reconstruction," *arXiv preprint*, 2023.

[12] J. He, Q. Xu, Y. Jiang, Z. Wang, and Q. Huang, "Ada-gad: Anomaly-denoised autoencoders for graph anomaly detection," *arXiv*, 2023.

[13] Y. Liu, Z. Li, S. Pan, C. Gong, C. Zhou, and G. Karypis, "Anomaly detection on attributed networks via contrastive self-supervised learning," *IEEE TNNLS*, 2021.

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

[15] 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.

[16] B. Chen, J. Zhang, X. Zhang, Y. Dong, J. Song, P. Zhang, K. Xu, E. Kharlamov, and J. Tang, "GCCAD: Graph contrastive learning for anomaly detection," *IEEE TKDE*, 2022.

[17] J. Duan, S. Wang, P. Zhang, E. Zhu, J. Hu, H. Jin, Y. Liu, and Z. Dong, "Graph anomaly detection via multi-scale contrastive learning networks with augmented view," in *AAAI*, 2023.

[18] Z. Liu, C. Cao, F. Tao, and J. Sun, "Revisiting graph contrastive learning for anomaly detection," *arXiv*, 2023.

[19] J. Pan, Y. Liu, Y. Zheng, and S. Pan, "Prem: A simple yet effective approach for node-level graph anomaly detection," *arXiv*, 2023.

[20] E. He, Y. Hao, Y. Zhang, G. Yin, and L. Yao, "Scala: Sparsification-based contrastive learning for anomaly detection on attributed networks," *arXiv*, 2024.

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

[22] X. Wang, B. Jin, Y. Du, P. Cui, Y. Tan, and Y. Yang, "One-class graph neural networks for anomaly detection in attributed networks," *Neural Computing and Applications*, vol. 33, no. 18, pp. 12 073–12 085, 2021.

[23] Y. Zheng, M. Jin, Y. Liu, L. Chi, K. T. Phan, and Y.-P. P. Chen, "Generative and contrastive self-supervised learning for graph anomaly detection," *IEEE TKDE*, 2021.

[24] Z. Liu, C. Cao, and J. Sun, "Mul-gad: a semi-supervised graph anomaly detection framework via aggregating multi-view information," *arXiv*, 2022.

[25] R. Fathony, J. Ng, and J. Chen, "Interaction-focused anomaly detection on bipartite node-and-edge-attributed graphs," in *IJCNN*. IEEE, 2023.

[26] 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.

[27] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *WWW*, 2019.

[28] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *KDD*, 2019.

[29] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," *NeurIPS*, 2019.

[30] X. Fu, J. Zhang, Z. Meng, and I. King, "MAGNN: Metapath aggregated graph neural network for heterogeneous graph embedding," in *WWW*, 2020.

[31] X. Yang, M. Yan, S. Pan, X. Ye, and D. Fan, "Simple and efficient heterogeneous graph neural network," in *AAAI*, 2023.

[32] 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.

[33] S. Zhu, C. Zhou, S. Pan, X. Zhu, and B. Wang, "Relation structure-aware heterogeneous graph neural network," in *ICDM*. IEEE, 2019.

[34] H. Hong, H. Guo, Y. Lin, X. Yang, Z. Li, and J. Ye, "An attention-based graph neural network for heterogeneous structural learning," in *AAAI*, 2020.

[35] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *The Web Conference*, 2020.

[36] Q. Lv, M. Ding, Q. Liu, Y. Chen, W. Feng, S. He, C. Zhou, J. Jiang, Y. Dong, and J. Tang, "Are we really making much progress? revisiting, benchmarking and refining heterogeneous graph neural networks," in *KDD*, 2021.

[37] R. G. Iyer, W. Wang, and Y. Sun, "Bi-level attention graph neural networks," in *ICDM*, 2021.

[38] Q. Mao, Z. Liu, C. Liu, and J. Sun, "Hinormer: Representation learning on heterogeneous information networks with graph transformer," in *WWW*, 2023.

[39] 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.

[40] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," 2014.

[41] 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.

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

[43] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: user movement in location-based social networks," in *KDD*, 2011.

[44] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," *IBM*, 1966.

[45] G. Niemeyer, "Geohash," *Geohash*, 2008.

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

[47] 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.

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

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

[50] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *NeurIPS*, 2019.

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

[52] R. Bing, G. Yuan, M. Zhu, F. Meng, H. Ma, and S. Qiao, "Heterogeneous graph neural networks analysis: a survey of techniques, evaluations and applications," *Artificial Intelligence Review*, 2023.