An Attributed and Diverse Encoder-Decoder Processing Technique for Anomaly Detection.

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Kenneth Antony John

December 2024

The Designated Project Committee Approves the Project Titled

An Attributed and Diverse Encoder-Decoder Processing Technique for Anomaly Detection.

by

Kenneth Antony John

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

December 2024

Katerina Potika	Department of Computer Science
Navrati Saxena	Department of Computer Science
William Andreopoulos	Department of Computer Science

ABSTRACT

An Attributed and Diverse Encoder-Decoder Processing Technique for Anomaly Detection.

by Kenneth Antony John

Attributed graphs are graphs that contain extra information about the attributes of nodes and edges. They can be used to model a plethora of real-world scenarios like social networks, bank transactions, and even academic citation data. Anomalies in such graphs can be irregularities or unusual patterns that are observed in the attributes or the structure of the graph. Anomaly detection in attributed networks is a crucial task, aiming to identify such anomalies. Existing methodologies use various deep learning techniques using graph neural networks, graph encoder-decoder architectures, and multi-layer perceptions. This study proposes a new approach to improve the existing methods using different types of neural networks. Additionally, it takes into account the different type of attributes (relations) on edges that are present in some datasets that contain real-world anomalies, like the DGraph-Fin. The experiments are performed on datasets with synthetic and real anomalies. We observed that the density of the graph affected the efficiency of the different types of Graph Neural Networks. We provide experimental results on four datasets and four different Graph Neural Network approaches and show that we have the same or improved precision and recall.

Keywords— Graph Anomaly Detection, Neural Networks, Graph Autoencoder

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my faculty advisor, Prof. Katerina Potika, for her invaluable guidance, patience, and support throughout this research journey. Prof. Potika's expertise and insightful critiques have significantly shaped this work.

My sincere thanks also go to my committee members, Prof. Navrati Saxena and Prof. William Andreopoulos, whose expertise and constructive feedback were instrumental in refining my research objectives and methodologies. Their perspectives and suggestions have greatly contributed to this study, improving it in many ways.

I am grateful for the opportunity to work under the guidance of such dedicated and knowledgeable mentors. Thank you for investing your time and effort in me and my work. It has been an honor to learn from each of you.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Problem Definition

Graphs can be used to represent entities as nodes and their relations as edges. They are commonly used in various domains like social networks, bank transactions, and communication networks. Graph anomaly refers to an irregular or unique pattern within a graph that deviates significantly from the rest of the dataset.

Anomalies in graphs can be observed in different forms like unusual cliques, unexpected connections between nodes, or distinctive attributes of individual nodes. In the real-world these graphs can represent financial transactions, computer networks, or even a set of people going to the same gym. Detecting such anomalies is crucial as they often indicate significant, events such as fraud in financial transactions, intrusion in network security, or disease spread in a community. The identification of graph anomalies relies on algorithms that bring out unique characteristics, that stand out from the typical behaviors observed in the graph, thereby highlighting outliers in the graph.

Consider a sample graph (Figure 1) where the nodes represent individuals in a social network, with attributes like gender and age where M and F denote if the person is Male or Female and the number next to it denotes the age of the person. In this context, a node anomaly could be an individual whose attributes or connections are significantly different from others within the same network. For example, a younger individual with an unusually high number of connections outside their typical age group could be considered as an anomalous node. This example emphasizes the importance of effectively detecting such anomalies, as they can highlight critical insights or potential areas of concern within the graph's structure.

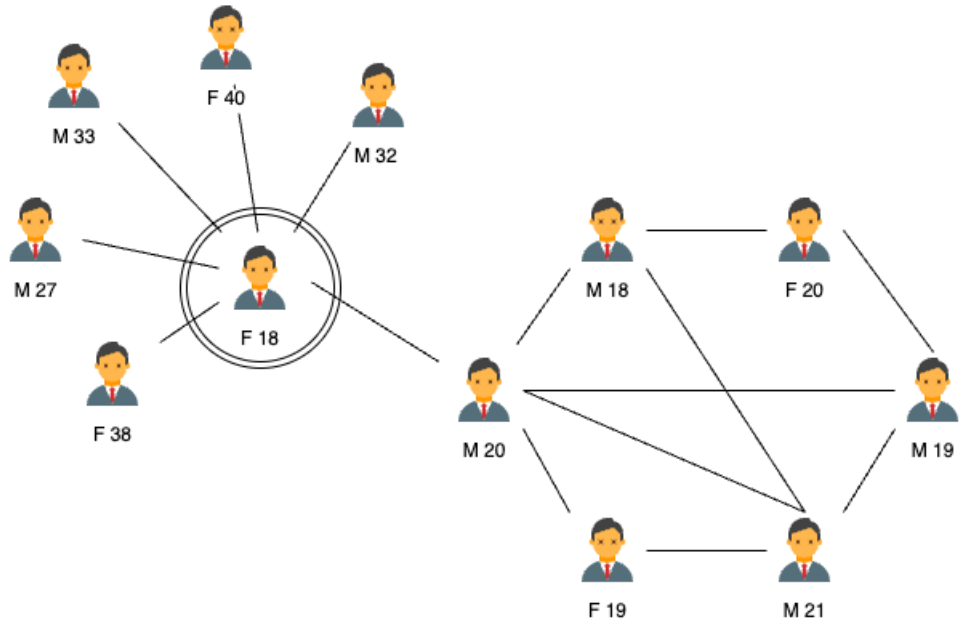


Figure 1: Attributed graph with anomalous node

1.2 Motivation

A$\hat{A}$ Anomaly detection in graphs is an important area of research with various real-world applications. This subsection dives deep into the various domains where anomaly detection plays a fundamental role, showcasing its critical importance. Some examples include:

- In the world of financial networks, the relationships between entities like banks, user accounts (nodes), and transactions (edges) can be modeled using a graph structure [1]. Detecting anomalies in such networks can be used to find out fraudulent activities. These activities range from money laundering to unusual transactions that deviate from the norm, presenting a clear danger to financial security.

- In computer networks, where the devices (nodes), and the wired or wireless connections (edges) form a graph, anomalies within these graphs can indicate unauthorized access, the propagation of malware, or other forms of unusual behavior [2]. These

anomalies are not merely disruptions to the network, but rather potential gateways for attackers. Early detection and mitigation of such anomalies are important for maintaining cybersecurity.

- The healthcare sector also benefits from anomaly detection in graphs, especially in monitoring disease spread due to physical interaction (edges) among people (nodes) who are affected by the disease. Here graphs can model the dynamics of disease transmission among populations [3]. Anomalies in these graphs could signal unexpected disease outbreaks and detection of such anomalies will help in finding out the source and ultimately stop the spread of the disease to other communities.

- Lastly, in the context of energy grid monitoring, graphs model the connections between power stations, substations, and consumers (nodes) connected by transmission lines (edges) [4]. Anomalies detected in these graphs can highlight faults in the energy grid, potential failures, or abnormal patterns of energy consumption. Such insights are valuable for preventing potential failures and ensuring the stability and efficiency of energy distribution systems.

Through these examples mentioned above, it becomes evident that anomaly detection is a necessity across a diverse range of applications. From safeguarding financial networks to ensuring the robustness of critical infrastructure, the role of anomaly detection is crucial.

This research proposes an attributed and diverse encoder-decoder processing technique abbreviated as **ADEPT** to tackle the problem of anomaly detection in attributed graphs. **ADEPT** uses the deep learning architecture of an encoder-decoder to create latent representations of the nodes in an attributed graph containing anomalies and then reconstruct the graph. The intuition behind this technique is that, as there is a class imbalance the number of anomalous nodes will be significantly lesser than the normal nodes. Hence the reconstruction error of the anomalous nodes will be greater than the normal nodes. This error can be used to rank the anomalous nodes. The type of neural network used for the

encoder and decoder can be tailored to the different kinds of graphs that needs anomaly detection.

This research is structured as follows: Chapter 2 defines some of the technical terms that are going to be used throughout the research. Chapter 3 describes some of the existing research in the field of anomaly detection in attributed networks. Chapter 4 focuses on the proposed methodology **ADEPT** and the experimental setup of the research, and Chapter 5 lays down the experimental results of this research and compares them with the existing results. Finally, Chapter 6 summarizes the results and why certain types of neural networks work better than others.

CHAPTER 2

Terminology

2.1 Graphs

A graph can be defined as a mathematical structure $G = (V, E)$ consisting of two sets V (vertices) and E (edges) [5]. Each edge can be connected to one or two vertices.

- V is a set of vertices, with n number of vertices or nodes
- $E \subseteq \{(u, v) \mid u, v \in V\}$ is a set of edges, with m number of edges

Each edge is a pair $e = (u, v)$, which denotes that an edge exists between u to v.

2.1.1 Undirected Graphs

An undirected graph will always be symmetric, i.e. undirected graphs can represent symmetric relationships. Take Figure 2 as an example where each node represents a person in a friend group and each edge between two nodes denotes if the two people are friends with each other b . C is connected with B which also means that B is connected with C.

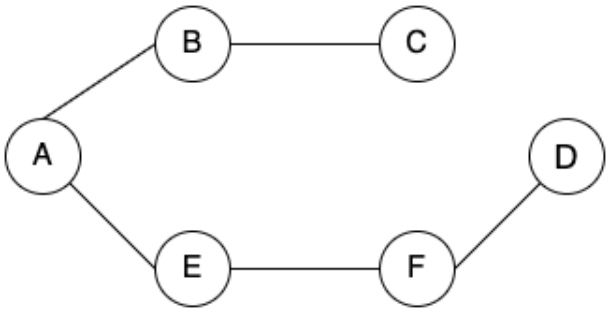


Figure 2: Undirected graph

2.1.2 Directed Graphs

A directed graph or a digraph has its edges represented using ordered pairs $e = (u, v)$, which denotes that a directed edge exists from u to v. A digraph can also have self-loops $e = (u, u)$. Directed graphs can be used to represent asymmetric relationships as shown in Figure 3, representing letters sent across towns in a state. As an example each town is represented using nodes and an edge from B to C denotes that letters have been sent from town B to town C.

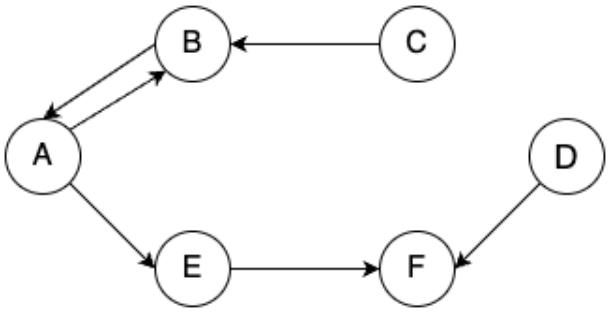


Figure 3: Directed graph

2.1.3 Graph Density

Graph density is the property of a graph that provides insights into how sparse or dense a graph is. The density value ranges from 0 to 1; 0 indicates that no edges are connected while a density of 1 indicates that it is a complete graph. Graph density can be calculated using the ratio of the number of edges in the graph to the maximum possible number of edges. The formula to calculate density differs for undirected and directed graphs.

$$\text{Undirected Graph Density} = \frac{2m}{n(n-1)} \qquad \text{Directed Graph Density} = \frac{m}{n(n-1)}$$

where:

n = number of nodes

m = number of edges

2.1.4 Heterogeneous Graphs

Heterogeneous graphs are graphs which have different types of nodes or edges. Formally, a graph is heterogeneous if the number of types of nodes or edges is greater than 1 [6]. A real-life example of a heterogeneous graph is an online retail platform like Amazon, where nodes represent entities such as products, users, and categories, and edges reflect relationships like purchases and product-category memberships. Figure 4 shows one such graph where the nodes can be of the type *user*, *products*, or *categories*, represented using the different shapes of nodes, and the edges can be of the type *purchases* or *belongs-to*, represented using the text above each directed edge. In this research, we will be focusing on one such special type of heterogeneous graph with a single type of node and multiple types of edges.

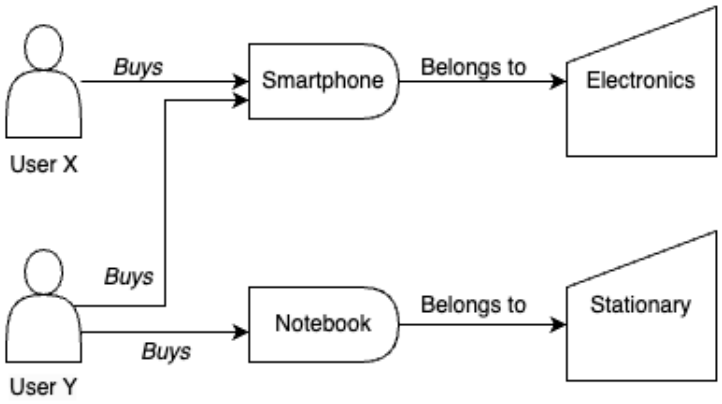


Figure 4: Heterogeneous graph

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 5: Sparse Adjacency matrix

2.1.5 Adjacency Matrix

The graph depicted in Figure 3 can be represented using an adjacency matrix. The adjacency matrix for a graph with n vertices will be of size $n \times n$. Consider an $n \times n$ matrix with each row i and each column j representing the nodes of a graph. An edge from the i^{th} node to the j^{th} node can be represented with a **1** in the matrix in the cell $[i, j]$ and the absence of an edge can be represented with a **0** in the respective cell.

2.1.6 Sparse Adjacency Matrix

Sparse matrices are used when the number of edges is significantly lesser than the number of nodes in the graph, i.e., the majority of the elements in the adjacency matrix are **0**s, see Figure 5. Here, only the non-zero entries are stored along with their positions. This saves up space required to store the adjacency matrix of large graphs and makes matrix operations like matrix multiplication efficient. One method of representing a sparse adjacency matrix is by using the Compressed Sparse Column (CSC) format. A matrix is represented by 3 main vectors that capture the non-zero elements, their row indices, column indices, and the weights of the respective edge (Figure 6). Figure 6a represents the vector consisting of the row indices of each node and Figure 6b represents the vector consisting of the column indices of all the nodes. For example, if the 0^{th} node is connected to the 1^{st} node, then the row indices vector will have 0 and the column indices vector will have 1 in the same position. As we are working with unweighted edges, the values of the third vector will always contain **1** and this vector representing the weights is not necessary and hence,

$$\begin{bmatrix} 0 & 0 & 1 & 2 & 3 & 5 \end{bmatrix} \qquad \begin{bmatrix} 1 & 4 & 0 & 1 & 5 & 5 \end{bmatrix}$$

(a) Row indices (b) Col indices

Figure 6: Sparse Adjacency Matrix in CSC format

not shown in the figure.

2.1.7 Clique

A clique is defined as a sub-graph g in graph G. where all the pairs of distinct nodes (u, v) are connected with an edge [7]. Figure 7 represents a graph G wherein the subgraph g, consisting of the nodes A, B, E, and F forms a clique as all the nodes in g are interconnected with each other.

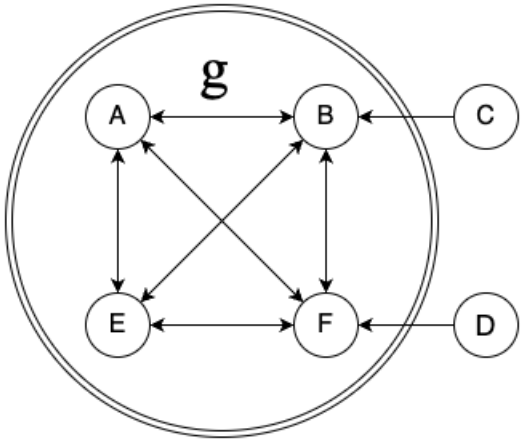


Figure 7: Clique in a graph G

2.2 Deep Learning Methods

This section talks about the deep learning techniques that are discussed and implemented throughout this research.

2.2.1 Graph Neural Network

Graph Neural Networks (GNNs) [8] represent a class of neural network models specifically engineered for processing graph-structured data. This approach allows GNNs to utilize both node features and the structure of the graph, enabling the learning of representations that capture the relational dependencies and patterns inherent in the data. The versatility of GNNs is evident in their wide range of applications across various tasks on graphs, including but not limited to node classification, link prediction, graph classification, and recommendation systems. The development and refinement of GNN architectures have led to the emergence of several notable models, some of which will be discussed below. Each of these architectures solves specific problems related to graph data.

2.2.2 AutoEncoder

Graph AutoEncoders [9] is an unsupervised learning process that takes advantage of unlabelled data and learns information about the data that can be used in another context. An autoencoder has 2 parts; an Encoder and a Decoder. First, the encoder compresses the input data into a lower dimensional latent space. The decoder attempts to recreate the original input from the compressed representation of the input data from the output of the encoder, aiming to preserve as much of the original information as possible. The difference between the recreated input and the original input is called the reconstruction error. This error serves as a feedback mechanism to improve the encoder and the decoder's performance using multiple iterations.

The capability of autoencoders to learn efficient representations without the need for labeled data makes them very useful for a variety of applications. For instance, they are commonly used in anomaly detection, where the learned representations can help identify outliers in the data by picking out instances with significantly higher reconstruction errors. In the field of image processing, autoencoders are employed for noise reduction, image com-

pression, and even in generative models, where they can create new images similar to the ones they were trained on.

2.3 Anomalies

According to the 2009 survey by Chandola et. al. [10], "anomalies are patterns in data, that do not conform to a well-defined notion of normal behavior." Anomalies are different from noise in the dataset, as noise is concerned with the data which is not relevant but is undesired. Whereas anomalies are interesting and relevant as they can denote various scenarios like fraud, network intrusion, disease outbreak, or grid failures.

2.3.1 Graph Anomaly

Graph anomalies are defined as the nodes, edges, or sub-graphs that appear to deviate significantly from other members of the graph in which it occurs [11] [12]. This research will be focused on anomalous nodes. Anomalous nodes can occur due to unique attributes or unexpected edge connections in the graph. For example, a person (node) with a certain strain of a virus in their bloodstream in close contact (edge) with a group of healthy people could be considered a nodal anomaly. Whereas, a transaction (edge) of $1000 when the average transaction from that particular bank account (node) is around $50 can be considered as an edge anomaly.

2.3.2 Anomaly Score

An anomaly score is a metric that can be used to gauge how likely a certain node is more anomalous as compared to another node. A higher anomaly score would mean that a particular node is much more likely to be an anomaly than a node with a lower score. The reconstruction error from an autoencoder framework can be used to assign an anomaly score for each node. Since the autoencoder is mainly trained on normal data, when an anomalous data point is sent in as an input, the decoder recreates the data point with very low accuracy,

hence increasing the reconstruction error for that input. This reconstruction error can be synonymous with the anomaly score, where a higher anomaly score indicates that the input is anomalous. For example, if there is a graph that has 1000 nodes in which 5 of the nodes are anomalous, the graph would be better in reconstructing the normal nodes with more accuracy than the anomalous nodes, as the autoencoder was trained on majorly normal nodes. By sorting the output nodes in decreasing order we can find the top K anomalous nodes, where K is the threshold that can be decided by the user.

The next chapter talks about the existing related work that has been done in detecting anomalies in graph networks.

CHAPTER 3

Related Work

Various studies have been conducted to identify anomalies in attributed graphs. Anomalous node detection can be divided into 3 types: global anomalies, structural anomalies, and community anomalies [12]. Global anomalies are the nodes with deviated attributes in the graph; structural anomalies refer to the nodes that have unique structural properties when compared to the whole graph; and community anomalies constitute both the anomalous nodal and structural properties within the same community [13]. The existing methods for graph anomaly detection required tailoring the technique for a specific problem which demanded extensive domain knowledge. This section reviews graph anomaly detection using Graph Neural Networks (GNN).

Ding et al. [14] addresses the problems faced by shallow learning mechanisms used for anomaly detection in graphs like network sparsity and data nonlinearity issues. [14] proposed Deep Anomaly Detection on Attributed Networks (DOMINANT) which uses an autoencoder framework to detect global and structural anomalies in a graph network. The encoder, implemented using a GCN would take the attributed graph as an input and create node representations in a latent space. This is then passed on to the decoder framework which aims to recreate the structural and nodal attributes of the input graph. An anomaly score is assigned to each node after combining both the structural and attributal reconstructional errors.

One Class Graph Neural Network (OCGNN) [15] combines the representational ability of GNNs along with the traditional one-class objective function. The GNN aggregates the neighboring information of each node into embedding vectors in a lower dimensional latent space, and a hypersphere learning objective function minimizes the volume that encloses the embedding vectors. The nodes which fall outside the hypersphere is considered to be

anomalous.

Zhang et al. [16] aims to alleviate problems that noisy nodes bring in the autoencoder architecture and the potential problem of overfitting the normal and anomalous nodes by proposing, dual support vector data description-based autoencoder (Dual-SVDAE). Inspired by anomaly detection techniques [17, 15] which uses hypersphere learning objective functions, Dual-SVDAE takes a similar approach to learn the hypersphere boundary of the normal nodes in the latent space. An encoder is used to represent the node embeddings in a lower dimension. Then, two hypersphere learning mechanisms are employed to learn the structural and attribute properties of the graph. Finally, the anomalous nodes are identified by measuring the distance from the center of the learned hypersphere in both the structural and attributal latent spaces.

To further improve the performance of anomaly detection in attributed graphs, community-aware attributed graph anomaly detection framework (ComGA) [18] was proposed by Luo et al. to detect global and community anomalies. Along with the encoder-decoder architecture ComGA, uses a community detection module that propagates community-specific information of each node to the tailoredGCN (tGCN) layer using a gateway. The tGCN combines the structural, attributable, and community information of the nodes and forms a latent representation which is then decoded to find anomaly ranking.

Yuan et al. [19] proposed higher-order structure-based anomaly detection on attributed networks (GUIDE) to address the inability of normal autoencoder techniques to effectively utilize complex patterns among nodes to detect anomalies. This was implemented using a graph node attention layer which learns weights depending upon the difference in structure of a node and its neighbors.

In addressing the challenges of anomaly detection within attributed networks, the study by Peng et al. [20] introduces A Deep Multi-View Framework for Anomaly Detection on Attributed Networks (ALARM), a multi-view framework that manages the heterogeneous

characteristics of attribute space. Unlike traditional approaches that combine multiple views into a singular feature vector, which neglects the distinct statistical properties and complementary information of heterogeneous views, ALARM utilizes multiple graph encoders and a sophisticated aggregator to facilitate both self-learning and user-guided learning.

In enhancing anomaly detection within multi-view attributed heterogeneous networks, the research by Chen et al. [21] introduces anomaly on multi-view attributed networks on attributed networks (AnomMAN), a graph convolution-based framework that integrates the attention mechanism to analyze the significance of various views within networks. Generally, the GCN has the tendency to filter out high-frequency signals which may mean that the model is unable to capture the anomalies. AnomMAN implements a graph autoencoder module to convert the disadvantage of low-pass features into a beneficial attribute for identifying anomalies. First, the multiple views are represented as separate sub-networks and then the latent representations of the sub-networks are fused with an attention mechanism. Finally, the anomaly score is calculated using the reconstruction loss from the attribute and structural decoder.

Moving forward, Chapter 4 discusses the methodology overview and the project architecture detail.

CHAPTER 4

Methodology

This chapter contains a comprehensive overview of the processes and strategies used. The implementation plan outlines the step-by-step approach that is followed. Following this, we explore the datasets used, with specific details on how to clean and pre-process the data for experimentation. Finally, this chapter talks about the experimental setup highlighting the technologies and procedures followed to detect anomalies.

4.1 Implementation Plan

The plan of implementation for this project is mentioned below and is depicted in a flowchart in Figure 8. **A**ttributed and **D**iverse **E**ncoder-Decoder **P**rocessing **T**echnique or **ADEPT**, is the combination of multiple methods that can be used to detect and rank anomalies in attributed homogeneous or heterogeneous graphs. Figure 8 is explained in detail in this section.

1. In the case when the datasets do not have ground truth anomalies, structural and attribute anomalies need to be injected. This is achieved using the methodologies introduced in [22] and [23] by inducing graph structure and node attribute anomalies.

2. Once the dataset with anomalies is generated, it is fed into the AutoEncoder which attempts to encode the input into a smaller latent space and then re-creates the input. The experiment is done with three different types of encoders, to compare and contrast the performance within these three encoders. The types of encoders used are:

    - GCN (Graph Convolutional Network) [24]
    - GAT (Graph Attention Networks) [25]
    - GraphSAGE [26]

- RGCN (Relational Graph Convolution Network) [27]

3. Once the decoder regenerates the input to the best of its ability, it is then compared with the original input.

4. A loss function assigns a score based on the combination of the nodal and structural attributes of the reconstructed graph. The larger the difference between the original input and the reconstructed output, the higher the score.

5. The nodes are then ranked on the basis of decreasing scores.

6. This ranking is then tested by comparing it with the anomaly labels in the input graph.

4.2 Datasets

This section talks about the different real-world datasets that are used in the experimentation to find anomalous nodes in attributed graphs. Three of them (ACM, BlogCatalog, and Flickr) [28, 29] are commonly used in other research targeting the problem of anomaly detection in graphs. AMC, BlogCatalog, and Flickr are undirected graphs whereas, DGraph-Fin [30] is a directed graph. More details about the datasets used in this research can be found in Table 1.

- **ACM**

  The ACM dataset consists of academic research papers as nodes and their citations as edges. If paper A cites paper B, it is considered as an edge from $A \rightarrow B$. As this graph is undirected an edge from $A \rightarrow B$ also means that, an edge exists from $B \rightarrow A$. The attributes for each node are derived from the abstract section of the research paper.

- **BlogCatalog**

  BlogCatalog is a blog-sharing website. Users can follow each other to view blogs, and each user can have a set of interests or tags. The users are depicted as nodes in the

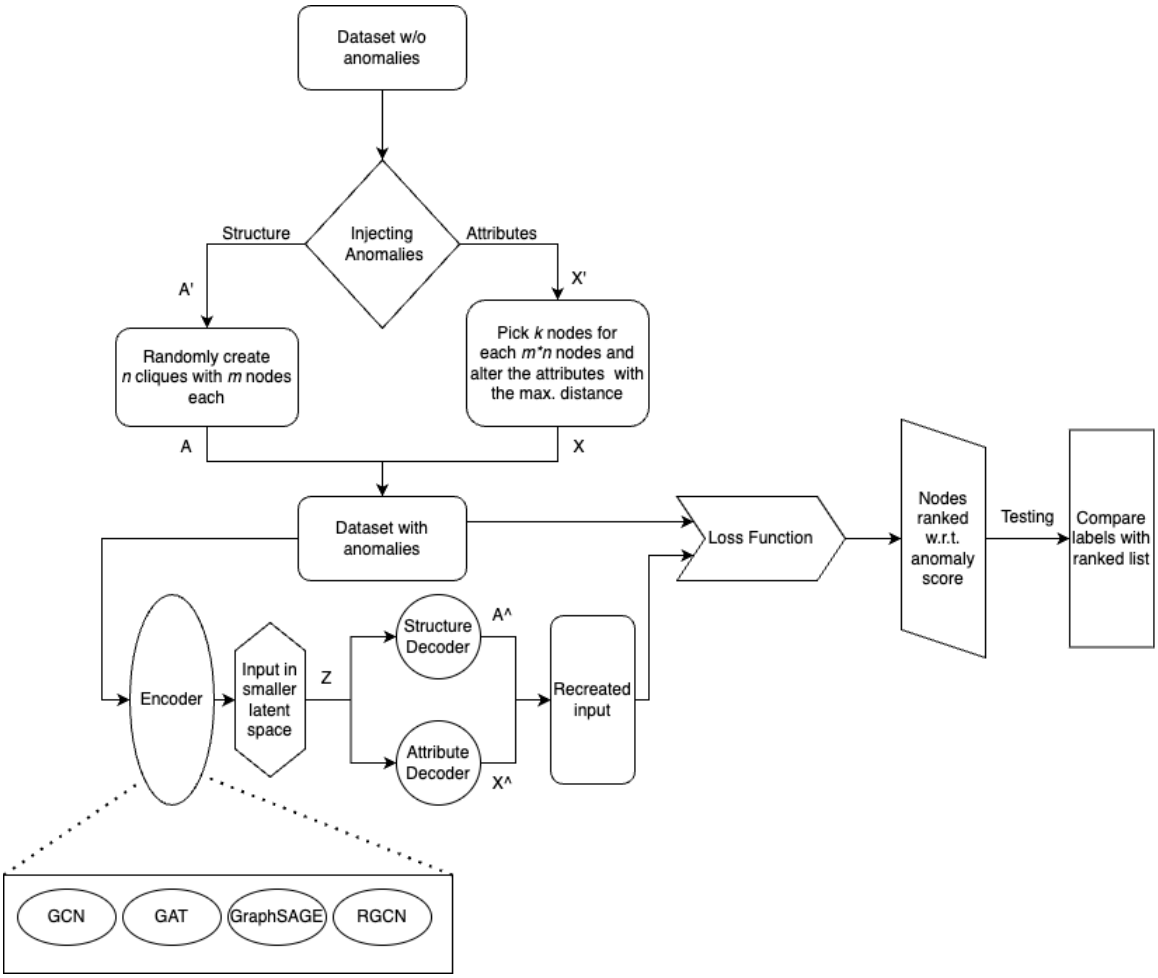


Figure 8: Workflow Diagram

Table 1: Dataset Information

Dataset	ACM	Flickr	BlogCatalog	DGraph-Fin
Total Edges	82,175	241,277	172,759	9718
Total Nodes	16,484	7,575	5,196	9000
Normal Nodes	15,887	7,130	4,898	8325
Anomalies	597	445	298	657
% of anomalies	3.62%	5.87%	5.73%	7.5%
Graph Density	0.06%	0.8%	1.2%	0.012%

graph and the tags for each user contribute to the attributes. Similar to ACM, the edges of the graph denotes each user following another user.

– **Flickr**

Flickr is an image-sharing website. The graph is formed by the users following each other and the nodal attributes are a combination of the user's interests and the tags associated with the images uploaded. These tags can capture details like location, image dimensions, and other camera configurations.

– **DGraph-Fin**

DGarph-Fin represents a real-world social network in the financial industry [30] from the fintech, Finvolution Group. A node represents a user, and an edge from user A to user B would mean that user A considers B as an emergency contact. The different types of edges in this dataset denote the sub-groups in the types of emergency contacts. This directed graph in itself has anomalies labeled within, and hence anomaly injection is not performed on this dataset.

4.3 Anomaly Injection

The following methods are used to inject structure and attribute anomalies into the datasets that do not have anomalies.

- Graph Structure -

  The graph structure is altered by adding small cliques to the graph [22]. This logic is backed by the fact that small cliques are frequently anomalous substructures in real-world situations. A small number of nodes that are more tightly linked to one another than the rest of the graph certainly does stand out to be anomalous. This is achieved by randomly selecting x nodes from the graph and then connecting them with each other by adding edges. This is done y times, resulting in y cliques with x nodes each. All these $x \times y$ nodes are labeled as anomalous nodes.

- Nodal Attributes -

  The attributes of each node in the graph are altered by the methods used in [23]. To make sure that the structure and attribute anomalies are the same, we pick $x \times y$ nodes in random. For each node k in $x \times y$, n nodes are picked randomly, and the node m, which has the most deviation in the feature vector is chosen. This deviation is calculated using the Euclidean distance between both of the node's feature vectors. Then the attributes of k are then altered to that of m.

4.4 Data Preprocessing

Data Preprocessing [31] is the method used to clean the datasets from noise or inconsistent data. It is also used to transform the data into useful forms that the model can use by performing consolidating or aggregating operations on the data. This section talks about the data preprocessing steps that were taken before the attributed graphs were fed into the model architecture.

4.4.1 Data sampling

As the core structure of the DGraph-Fin[30] dataset is different than that of the rest of the datasets used in this research, additional data sampling methods are used for the

DGraph-Fin dataset. The original dataset has 3,700,550 nodes and 4,300,999 edges. For the purpose of this research, the size of the dataset had to be reduced to align with the technical limitations. A total number of 9000 nodes were chosen from the training mask provided in the dataset. After multiple iterations of randomly picking nodes, one such set with 657 anomalous nodes was selected for this research. This selection was made to closely represent the structure of the original dataset as much as possible using Algorithm 1.

The Algorithm accepts *totalAnomalies*, which is the number of expected anomalies, *totalNodes* which is the total number of nodes that we are aiming for this sampled data, and the *data* object which contains 17-dimensional node features, *nodeLabels* where 1 meant that the node was anomalous, *edgeIndex* which denotes the edges of the graph, *edgeTypes* which indicates the type of the edge for each edge, and *filteredNodesMask* which is a random split of 70% of the nodes from the whole dataset.

Algorithm 1 randomly picks out nodes from the input graph with a certain number of total nodes and anomalous nodes to replicate the original dataset. The parameters *totalAnomalies* and *totalNodes* decide the number of nodes. For this research, the total number of nodes was set to 9000 and the number of anomalous nodes was set to 657.

Algorithm 1 Data Sample DGraph-Fin

procedure INITIALIZE(DATA, TOTALANOMALIES, TOTALNODES)

label ← data.label ▷ boolean array where true = anomaly
filteredNodesMask ← data.filteredNodesMask
▷ boolean array; training data from dataset

combinedNodes ← filteredNodesMask ∩ label

if len(combinedNodes) < totalAnomalies elements **then**
    raise error "Not enough True values to select k"

selectedAnomalies ← *RandomSelect*(combinedNodes, totalAnomalies)

anomalyMask ← [False] * *shape*(filteredNodesMask)

anomalyMask[selectedAnomalies] = True

filteredNodes ← {i : filteredNodesMask[i] = True}

eligibleCandidates ← {x : x ∈ filteredNodes and x ∉ anomalyMask}

if len(eligibleCandidates) < totalNodes **then**
    raise error "Not enough True values to select required amount"

possibleCandidates ← *RandomSelect*(eligibleCandidates, totalNodes)

selectedNodes ← selectedAnomalies ∪ possibleCandidates

selectedNodesMask ← [False] * *shape*(filteredNodesMask)

Set selectedNodesMask[selectedNodes] = True

return selectedNodesMask

4.4.2 Normalization of Adjacency Matrix

For the graphs, the adjacency matrix was symmetrically normalized using the steps mentioned in Algorithm 2. This is a crucial step during pre-processing as it helps to contain the values of the matrices when it undergoes multiple multiplication operations.

Algorithm 2 Normalize Adjacency Matrix [32]

```
1: procedure NormalizeAdj(A)
2:     A ← toCoordinateFormat(A)
3:     rowSum ← sum of each row of A
4:     sqrtRowSum ← √row for each row in rowSum
5:     for each row in sqrtRowSum:
6:     if sqrtRowSum[row] = ∞ then
7:         sqrtRowSum[row] = 0
8:     diagMatInvSqrt ← getDiagonalMat(sqrtRowSum)
9:     normalizedA ← (adj × diagMatInvSqrt)ᵀ × (diagMatInvSqrt)
10:    return normalizedA
```

The functions **toCoordinateFormat()** and **getDiagonalMat()** are not being elaborated further to maintain brevity. These functions come out of the box with the **scipy** [33] library. The function, **toCoordinateFormat()** converts the input from *csc_ matrix* format to *coordinate_ list* format and the **getDiagonalMat()** creates a diagonal matrix depending on the input.

4.5 Experimental Setup

This section talks about the various modules present in this project and how they all tie together to achieve our goal of anomaly detection as shown in Figure 9. The flow starts from the preprocessed data being fed into the encoder as the input and a latent representation of the data is produced as the output. The encoder is optimized to retain as much information from the input graph within this representation. This information in a lower dimension space is then taken in as the input of the decoder. The decoder then attempts to recreate the structural component and feature attributes of the input graph.

Once the structural component and the attributes are reconstructed, represented using an adjacency matrix and a feature matrix, a loss function is applied. This function compares the initial input and the reconstructed matrices and comes up with a score to rank the nodes based on the anomalies.

Once this ranking is done, the performances of the multiple architectures can be evaluated using metrics like Area Under the ROC Curve (AUC), Precision, and Recall.

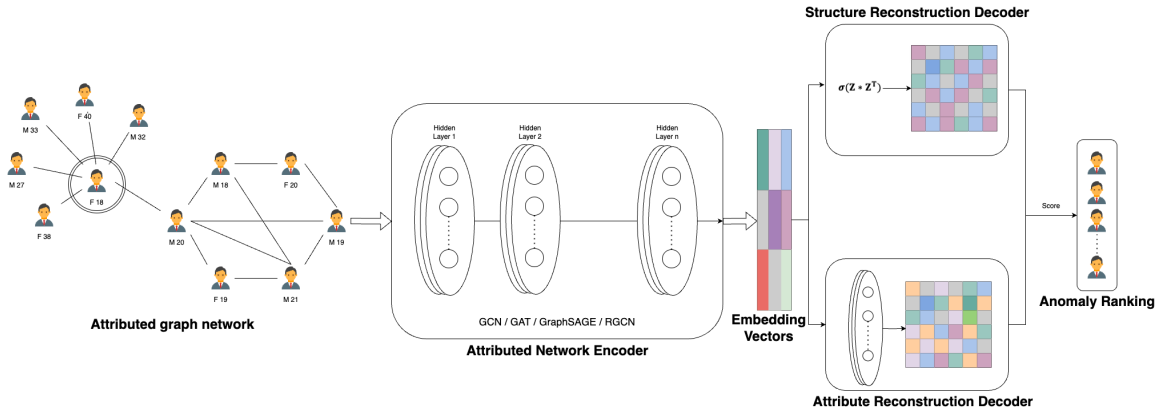


Figure 9: Architecture Diagram For ranking anomalous nodes

4.5.1 Encoder

The encoder is responsible for taking in the structural and nodal attributes as the input to then convert it into a latent representation in a lower dimension. This module tries to capture the information of the graph as efficiently as possible, without significant information loss.

For this research study, 3 different encoder architectures were implemented with different settings to compare and contrast their respective performances with each other:

1. **GCN**

   Graph Convolutional Networks (GCNs) [24] are a specialized category within the framework of GNNs, distinguished by their innovative use of convolutional operations to facilitate information propagation among nodes within a graph. These networks harness the power of localized aggregation, where features of neighboring nodes are aggregated to update node representations. The GCN architecture comprises of multiple layers that perform a graph convolution operation, which is followed by the application of a non-linear activation function to further enhance the model's learning

capability. This methodology has proven effective in various tasks, most notably in node classification. In such tasks, GCNs perform well labeling nodes using a combination of the graph's structural information and their attributes. This highlights the promise of GCNs in gaining valuable insights from graph-structured data.

The architecture of the GCN was inspired by the works of Kipf [24] et al. This was the architecture that was used in DOMINANT[14]. The multi-layer GCN follows the propagation rule as mentioned in:

$$H^{(l+1)} = ReLU\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{1}$$

Here, $\tilde{A}$ is the adjacency matrix with self-loops added in. $\tilde{D}$ is a diagonal matrix where each of the i^{th} element is the sum of each row in $\tilde{A}$. $H^{(\text{l})}$ is the input for the l^{th} layer and $H^{(\text{l}\,+\,1)}$ is its output. For the 1^{st} layer, the feature matrix is taken in as the input. $W^{(\text{l})}$ is the weight matrix for the l^{th} layer which is learned through the convolution process.

For this research, 3 GCN layers were implemented with a dropout of 30%

2. **GAT**

   Graph Attention Network (GAT)[25] is an advancement in GNN architectures, that integrates attention mechanisms to emphasize significant node relationships. In this approach, GAT allocates attention weights to the neighbors of a given node to prioritize the important ones during the aggregation process. GAT fine-tunes the aggregation of relevant information through the computation of attention coefficients based on the target node's and neighbor's feature vectors. Different graph relationships can be computed using the multi-head functionality of GAT, where each head computes the attention weights and aggregates the node feature embeddings. GAT architectures are widely used for tasks like node classification, link prediction, and graph classification, all of which require effective handling of complex relational data.

GAT [25] is used as a substitute for the boilerplate GCN from the previous method. The use of GATs offers several advantages over GCNs in handling graph data. GATs dynamically learn the relative importance of each node's neighbors, this enables efficient management of heterophilic graphs[34]. Heterophilic graphs are graphs, where connected nodes may not have similar features—not to be confused with heterogeneous graphs where there are different types of edges. This capability enhances the model's capacity to distinguish subtle and complex patterns within the graph. This is specifically important in anomaly detection as the graph might be heterophilic in nature, and hence GAT has an upper hand over GCN for our task.

Furthermore, GATs improve feature learning through weighted contributions from each of the node's neighbors. This allows for the effective amalgamation of neighbor attributes, which enhances the overall predictive performance of the model.

In a single layer of GAT, the computation of attention coefficients for a node i involves 4 important steps. First, a liner transformation is applied to the input features of the node and its neighbors. Then the attention coefficients for the node and its neighbors are calculated. These attention coefficients are used as weights to aggregate the features of the neighbors. Finally, a non-linear activation function is applied to the aggregated features.

The attention coefficients α_{ij} between node i and node j are calculated as:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\vec{a}^T[\mathbf{W}\vec{h}_i \parallel \mathbf{W}\vec{h}_j]))}{\sum_{k\in\mathcal{N}(i)} \exp(\text{LeakyReLU}(\vec{a}^T[\mathbf{W}\vec{h}_i \parallel \mathbf{W}\vec{h}_k]))} \tag{2}$$

Where $\vec{h}_i$ is the feature vector of node i, $\mathbf{W}$ is a learnable weight matrix, $\vec{a}$ is a learnable attention vector, $\mathcal{N}(i)$ is the neighborhood of node i, and $\parallel$ denotes concatenation.

The new representation of the node i is given by:

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}\mathbf{W}\vec{h}_j\right) \tag{3}$$

Where σ is a non-linear activation function like ReLU. For this research 3 GAT layers

with ReLU as the activation function were used.

All the GNN architectures mentioned above were transductive algorithms, i.e. the whole graph should be trained to learn the node embeddings. So in the case of dynamic graphs, where new nodes are added to existing ones, it needs to be re-run to compute the node embeddings. Below we talk about one such GNN architecture which is different from the above.

3. **GraphSAGE**

   GraphSAGE [26] uses inductive learning, where it can predict the embedding of a new node without the need of re-training the model. It achieves this, by applying different aggregator functions that can be used to estimate the new node embedding. Furthermore, GraphSAGE's neighborhood sampling method samples a fixed size of neighbors for each node rather than all the nodes in the graph. This improves efficiency and decreases the compute needed leading to increased scalability. GraphSAGE is used for problems such as link prediction, node prediction, and network traffic flow.

   GraphSAGE offers significant advantages over GCNs, particularly in terms of scalability and flexibility. It effectively learns from large graphs due to its sampling strategy. The node-wise sampling method, samples neighbors with k depth from each node, where k can be configured using the number of layers of the model [35]. GraphSAGE is able to handle sparse and dense areas within the graph by sampling fixed-size neighborhoods. This is crucial for anomaly detection as most of the structural anomalies could be found in such areas. Its inductive learning capability allows it to generalize well for unseen nodes, which makes it a good choice for dynamic graphs.

   For each node, a fixed-size set of neighbors is sampled, and the neighbors' features are aggregated for each node. Then the aggregated neighborhood vector is concatenated with the node's own feature vector. This concatenated vector is then passed through a neural network later to produce the updated node embedding. The aggregation function used in this research is the mean aggregator. Here the element-wise mean of

the neighbors' feature vectors are taken to aggregate the features.

$$\mathbf{h}_v^{(k)} = \sigma\left(\mathbf{W}^{(k)} \cdot \text{mean}\left(\left\{\mathbf{h}_u^{(k-1)}, \forall u \in \mathcal{N}(v) \cup \{v\}\right\}\right)\right) \tag{4}$$

Where $\mathbf{h}_v^{(k)}$ is the embedding of node v at layer k, σ is a non-linear activation function like ReLU, $\mathbf{W}^{(k)}$ is the learnable weight matrix at layer k, mean$(\cdot)$ computes the element-wise mean of the feature vectors, $\mathbf{h}_u^{(k-1)}$ is the feature vector of node u from the previous layer $(k-1)$, $\mathcal{N}(v)$ is the set of neighbors of node v.

For this research 3 GraphSAGE layers with aggregation function as *mean* were implemented.

4. **RGCN**

Relational Graph Convolutional Networks (RGCNs) [27] are an extension of GCNs tailored for analyzing relational data, such as social networks or knowledge graphs. Unlike standard GCNs, RGCNs can handle various relationship types between nodes, which is crucial for datasets where connections between entities vary significantly. These networks operate by aggregating and updating node features through multiple convolution rounds, utilizing a learnable weight matrix and sometimes incorporating skip connections to retain information across these rounds. This feature aggregation process allows RGCNs to extract and learn intricate relational features effectively. RGCNs are widely applied in domains requiring detailed relationship analysis, including social network analysis, recommendation systems, and knowledge graph completion, where they enhance the prediction accuracy for links and entity classifications.

RGCNS has some key improvements over traditional GCNs, especially when dealing with heterogeneous graphs. RGCNs work well in modeling complex relational data by handling different types of edge relations separately. This allows RGCNs to capture patterns and relationships between nodes that might be missed by GCNs. Additionally, RGCNs incorporate edge-type information into the graph convolution process,

enhancing the model's ability to learn from the structural and semantic context of each node. This makes RGCNs best suited for tasks like link prediction in attributed heterogeneous graphs. This is useful for anomaly detection as link prediction is a crucial part of the decoder mechanism.

As there are multiple types of edges, each type of edge has its own weight matrix which is used to transform the features of the neighboring nodes. Then, for each node, the messages from its neighbors are aggregated, considering the type of edge through which the information is passed. The aggregated message is combined with the node's current representation to produce a new node representation. The representation of a node i at layer k is updated as follows.

$$\mathbf{h}_i^{(k)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}\mathbf{W}_r^{(k)}\mathbf{h}_j^{(k-1)} + \mathbf{W}_0^{(k)}\mathbf{h}_i^{(k-1)}\right) \tag{5}$$

Where $\mathbf{h}_i^{(k)}$ is the feature vector of node i at layer k, σ is a non-linear activation function like ReLU, $\mathcal{R}$ is the set of all possible edge types, $\mathcal{N}_i^r$ is the set of neighbors of node i connected by edge type r, $\mathbf{W}_r^{(k)}$ is the weight matrix for edge type r at layer k, $c_{i,r}$ is a normalization constant, $\mathbf{W}_0^{(k)}$ is the weight matrix for the self-loop. For the implementation, 2 RGCN layers were used, and the self-loops were removed before passing it on to the encoder.

4.5.2 Structure Decoder

The structure decoder attempts to re-create the adjacency matrix of the input graph from the latent representation. This is achieved by multiplying the latent representation with the transpose of itself and then running it through an activation function like ReLU (6). This computation's resultant matrix is considered the re-constructed adjacency matrix. The structure decoder is trained to predict if an edge exists for all the nodes in the graph.

$$\hat{A} = \mathbf{ReLU}(ZZ^T). \tag{6}$$

The reconstruction error for the structural properties of the graph can be calculated by matrix subtraction and then taking the distance between the two matrices as shown in (7).

$$R_{struct} = \|A - \hat{A}\|^2 = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(A_{ij} - \hat{A}_{ij})^2} \tag{7}$$

4.5.3 Attribute Decoder

The attribute decoder has the same structure as the Encoder mentioned in 4.5.1. The vital difference is that the decoder's input dimension will be that of the latent representation and the output dimension will be that of the original input graph. Similar to the encoder, multiple variations of the GCN are used in this research.

The decoder will output a node feature matrix for each node, and the reconstruction error is calculated using the Euclidean distance between the encoder input and the decoder output (8).

$$R_{attr} = \|X - \hat{X}\|^2 = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(X_{ij} - \hat{X}_{ij})^2} \tag{8}$$

4.5.4 Anomaly Ranking

As there is a class imbalance in the dataset, the whole Encoder–Decoder architecture is trained majorly on the normal nodes and it will perform well in reconstructing the normal nodes. In the case of anomalous nodes, it is expected to not perform well and hence it will have a higher reconstruction error.

This reconstruction error is a combination of both the structural and attribute errors

and is calculated using (9). The tunable parameter alpha (α) decides the weightage of each of the two errors. For example, when $\alpha = 0.4$, 40% of the weightage is given to the attribute reconstruction error and 60% is for structural reconstruction error.

$$Score = (1 - \alpha)R_{struct} + \alpha R_{attr} = (1 - \alpha)\|A - \hat{A}\|^2 + \alpha\|X - \hat{X}\|^2 \tag{9}$$

The nodes of the graph are then ranked on the basis of this *Score*, the higher the *Score* the higher the chance of the node being anomalous as compared to the roles below it.

4.6 System Specifications

All experiments were performed on an M1 MacBook Air, with the specifications of this machine detailed below. Table 2 highlights the key components, including the processor, memory, storage, graphics, and operating system. These specifications provide context for understanding the computational capabilities and limitations under which the research was carried out.

Specification	**Details**
Model	MacBook Air M1
Processor	Apple M1 chip, 8-core CPU
Memory (RAM)	16GB
Storage	256GB SSD
Graphics	Apple 8-core GPU
Operating System	macOS Ventura 13.5.2

Table 2: System Specifications

4.7 Hyper-parameter Tuning

Hyper-parameter tuning or hyper-parameter optimization is the process used to find the best performing hyper-parameters for a machine learning model. These parameters are not learned during the training process, but rather set prior to it and controls the behavior of the model. This process and significantly impact the performance of a machine learning

model and in finding a balance between under-fitting and over-fitting the data. There are multiple methods that can be used for hyper-parameter optimization.

4.7.1 Grid Search

For this research, a grid search has been performed for hyper-parameter tuning. This method involves specifying a finite set of values for each hyper-parameter and then trying all possible combinations. This process is exhaustive but can be computationally expensive [36]. As the results for one trial are independent of the others, grid search can be run in parallel to save time.

For this research, the parameters and its respective values tested for grid search are shown in Table 3.

Hyper-parameter	**Values Tested**
Hidden Layers (`no_of_layers`)	1, **2**, 3, 5
Dimensions of hidden layers (`hidden_dimensions`)	32, **64**, **128**, 256, 512, 1024
Maximum Epochs (`max_epochs`)	50, **100**, 300
Learning Rates (`lrs`)	0.003, **0.005**, 0.007
Dropout Rates (`dropout`)	0.1, 0.2, **0.3**, 0.4
Alpha (`alpha`)	0.6, 0.7, **0.8**, 0.9

Table 3: Hyper-parameters and their respective values used in grid search.

After performing grid search on the values mentioned in Table 3 the best-performing values were picked and the final values used for this research are as follows:

- Hidden Layers - 2

  The number of layers in between the input and the output layers helps in learning the structural and nodal properties of the graph.

- Dimensions of hidden layers

  The number of nodes or neurons in each of the layer.

  - Layer 1 - 64
  - Layer 2 - 128

- Maximum Epochs - 100

  The number of times the dataset is passed through the neural network.

- Learning Rate - $5 * 10^{-}3$

  The rate that determines the step size at each iteration when the model is trying to minimize the loss function

- Dropout Rate - 0.3

  The probability value of the number of neurons that are randomly dropped to prevent overfitting.

- Alpha - 0.8

  The tunable parameter that decides the weightage of each of the reconstruction errors.

CHAPTER 5

Experimental Results

This section presents the research results trained and tested on 4 different datasets and compares them with the various configurations of the Encoder-Decoder architecture discussed above. The performance of the proposed **ADEPT** models was evaluated based on evaluation metrics such as Precision and Recall. The benefits of each model in accurately ranking anomalous nodes in an attributed graph network are discussed and compared. These findings highlight how **ADEPT** could help in improving the ongoing research on anomaly detection in attributed graphs.

5.1 Evaluation Metrics

5.1.1 Precision

Precision talks about the proportion of correct anomalous identifications. This is the ratio of correctly identified anomalies over all the nodes that were identified as anomalous (10).

$$\text{Precision} = \frac{TruePositives}{TruePositives + FalsePositives} \tag{10}$$

High precision means that the model is good at avoiding false positives. In this case, a higher precision value would indicate that the model rarely misclassifies normal data nodes as anomalies. The calculated precision values for the top K nodes on each of the 4 datasets are shown in Table 4.

As it is clear in Table 4, using GAT instead of GCNs [14] as the encoder does improve the performance by around 5% for the Flickr dataset. Whereas using GraphSAGE works consistently better across the BlogCatalog dataset. For the heterogeneous dataset, RGCN performs significantly better than the rest of the methods used in the experimentation.

Table 4: Precision @ K

Dataset/Method	K	DOMINANT[14]	GAT	GraphSAGE	RGCN
ACM	50	0.68	**0.68**	**0.68**	-
	100	0.67	**0.67**	**0.67**	-
	200	0.66	**0.66**	**0.66**	-
	300	0.66	**0.66**	**0.66**	-
BlogCatalog	50	0.493	0.493	**0.497**	-
	100	0.54	0.54	**0.60**	-
	200	0.61	0.61	0.61	-
	300	0.7	0.7	0.7	-
Flickr	50	0.50	**0.62**	0.48	-
	100	0.67	**0.72**	0.67	-
	200	0.680	**0.705**	0.680	-
	300	0.627	**0.647**	0.623	-
DGraph-Fin	50	0.52	0.54	0.56	**0.64**
	100	0.56	0.54	0.56	**0.61**
	200	0.545	0.535	0.545	**0.58**
	300	0.533	0.53	0.53	**0.597**

5.1.2 Recall

Recall is the measure of the proportion of actual anomalies that were identified correctly. This is calculated by the ratio of correctly identified anomalies over all the actual anomalies in the dataset (11).

$$\text{Recall} = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{11}$$

A high recall value means that the model is good at identifying the majority of the anomalies in the dataset accurately, but it might also produce false positives. In our experimentation, we calculate the recall for the top K nodes as shown in Table 5.

Table 5 points out a similar trend in the precision and recall for the respective experimental settings. There is an improvement of around when GAT is used instead of GCN for Flickr. As expected the heterogeneous *FinGraph* dataset works well when a compatible

Table 5: Recall @ K

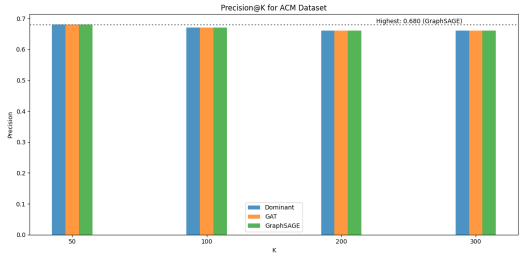
Dataset/Method	K	DOMINANT[14]	GAT	GraphSAGE	RGCN
ACM	50	0.613	**0.613**	**0.613**	-
	100	0.627	**0.627**	**0.627**	-
	200	0.657	**0.657**	**0.657**	-
	300	0.69	**0.69**	**0.69**	-
BlogCatalog	50	0.517	0.517	0.517	-
	100	0.605	0.605	0.605	-
	200	0.729	0.729	**0.73**	-
	300	0.791	0.791	**0.796**	-
Flickr	50	0.522	**0.536**	0.520	-
	100	0.583	**0.594**	0.583	-
	200	0.671	**0.682**	0.671	-
	300	0.720	**0.734**	0.718	-
DGraph-Fin	50	0.41	0.42	0.42	**0.506**
	100	0.45	0.43	0.43	**0.51**
	200	0.47	0.45	0.45	**0.514**
	300	0.48	0.47	0.47	**0.526**

RGCN is used.

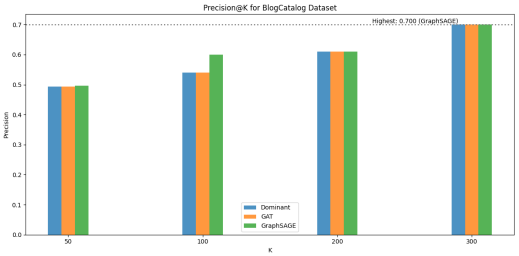
These results suggest that different configurations of the Encoder-Decoder architecture work well for certain datasets.

5.2 Inference

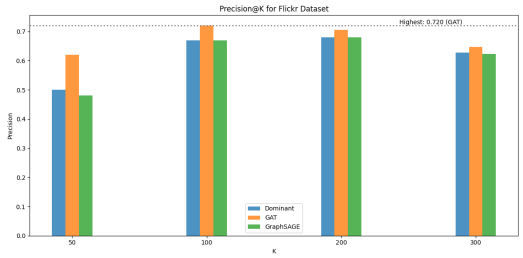
This section consolidates the results presented in the previous section and explains why different configurations in **ADEPT** work well for certain datasets. The experimental results reveal patterns in the performance of different graph neural network architectures across various datasets. These observations can be largely attributed to the varying graph densities and structural characteristics of the datasets used in the study. Figures 10 and 11 provide a holistic view of how each method works for each of the datasets and highlights the one that is performing the best.



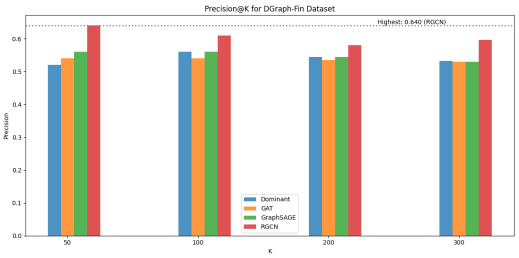
(a) ACM



(b) Blog Catalog

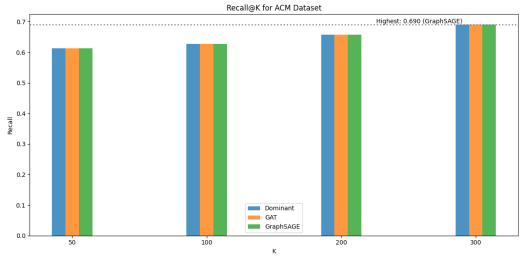


(c) Flickr

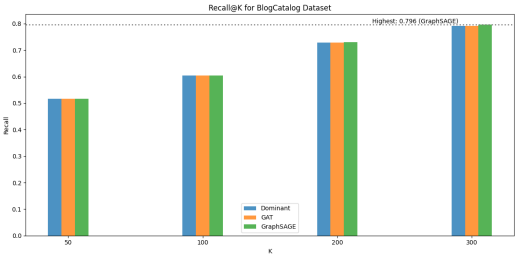


(d) DGraph-Fin

Figure 10: Precision across all datasets

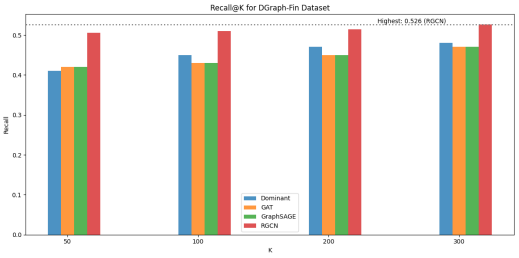


(a) ACM



(b) Blog Catalog

(c) Flickr



(d) DGraph-Fin

Figure 11: Recall across all datasets

The GAT encoder consistently outperformed the other methods on graphs with sparse densities such as Flickr (density = 0.8%). This can be attributed to GAT's ability to apply

attention mechanisms, allowing it to prioritize the most informative neighbors. By focussing on key relationships, GAT ensures that structural patterns are captured despite the lower edge density.

Whereas, GraphSAGE encoder exhibited better results on denser graphs like BlogCatalog (density = 1.2%). The inductive property of GraphSAGE allows it to aggregate information from a fixed-size neighborhood, making it suited for denser graphs. However, this may lead to information loss in sparser graphs where fewer nodes are available for aggregation.

For the heterogeneous DGraph-Fin dataset, the RGCN performed the best. This can be linked to RGCN's ability to model different types of relationships in the graph explicitly, which is very important for handling heterogeneous graph networks.

In summary, the performance of each of the architectures in **ADEPT** is highly influenced by the density and the heterogeneity of the graph. The GAT encoder works well with sparse graphs by focusing on the key links between nodes, while GraphSAGE performs better on denser graphs be aggregating neighborhood information effectively. For heterogeneous graphs, RGNC's ability to model different types of edges proves to be the most effective.

CHAPTER 6

Conclusions and Future Work

To conclude this research explored multiple ways to rank anomalies in attributed graphs. Attributed graphs can be found in a plethora of real-life scenarios such as bank transfer details, community graphs, and research paper citation networks. Anomalies in such graphs are defined as the elements in those graphs that are out of the norm from the rest of the dataset. This can be differentiated from noise, as noise in a dataset is attributed to the random errors that occur in the dataset. The reason why detecting such anomalies is important is that it indicates an underlying event in the dataset. For the examples mentioned above, it could mean a fraud transfer, disease outbreak, or a predatory journal.

The existing methods of detecting anomalies in attributed graphs explore machine learning methods using neural networks, multilayer perceptrons, and encoder-decoder architectures. This research focuses on the encoder-decoder methodology to improve and gain insights into the existing studies. This architecture compresses the graph into a latent space and then reconstructs them, to identify or rank anomalies based on the reconstruction errors. Using datasets commonly used in other anomaly detection research like BlogCatalog, Flickr, and ACM along with a recently published real world dataset DGraph-Fin provides a holistic view of how **Adept** compared to the rest of the methods.

Adept aims to determine how the type of GNNs used in the architecture affects the performance of the model for different kinds of datasets. The experiments performed reveal a relation between the graph density and the type of GNN used in the encoder-decoder architecture. GraphSAGE performed well for dense graphs due to its property of combining neighborhood information effectively, whereas GAT excelled in sparse graphs, as it focuses on the key edges between nodes. For heterogeneous graphs, RGCN seemed to be the best fit due to its ability to model various edge types.

As for future work in this area of research, multiple different datasets can be incorporated and tested on the existing implementation. This would provide more data points and hence, insights on how and why certain methods work well for certain datasets. Along with experimenting with newer datasets, other types of GNNs like MagNet, EGNN, PG-GNN [37] can be incorporated with the **ADEPT** architecture. These experiments will provide a deeper understanding and provide effective ways to detect anomalies in attributed graphs.

LIST OF REFERENCES

[1] W. Hilal, S. A. Gadsden, and J. Yawney, "Financial fraud: a review of anomaly detection techniques and recent advances," *Expert systems With applications*, vol. 193, p. 116429, 2022.

[2] J. Jiang, J. Chen, T. Gu, K.-K. R. Choo, C. Liu, M. Yu, W. Huang, and P. Mohapatra, "Anomaly detection with graph convolutional networks for insider threat and fraud detection," in *MILCOM 2019-2019 IEEE Military Communications Conference (MILCOM)*. IEEE, 2019, pp. 109114.

[3] V. La Gatta, V. Moscato, M. Postiglione, and G. Sperli, "An epidemiological neural network exploiting dynamic graph structured data applied to the covid-19 outbreak," *IEEE Transactions on Big Data*, vol. 7, no. 1, pp. 4555, 2020.

[4] A. Anwar and A. N. Mahmood, "Anomaly detection in electric network database of smart grid: Graph matching approach," *Electric Power Systems Research*, vol. 133, pp. 5162, 2016.

[5] P. Zhang and G. Chartrand, "Introduction to graph theory," *Tata McGraw-Hill*, vol. 2, pp. 21, 2006.

[6] Y. Sun and J. Han, "Mining heterogeneous information networks: a structural analysis approach," *ACM SIGKDD explorations newsletter*, vol. 14, no. 2, pp. 2028, 2013.

[7] R. D. Alba, "A graph-theoretic definition of a sociometric clique," *Journal of Mathematical Sociology*, vol. 3, no. 1, pp. 113126, 1973.

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

[9] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 7894, 2018.

[10] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 158, 2009.

[11] F. E. Grubbs, "Procedures for detecting outlying observations in samples," *Technometrics*, vol. 11, no. 1, pp. 121, 1969.

[12] 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 Transactions on Knowledge and Data Engineering*, 2021.

[13] H. Kim, B. S. Lee, W.-Y. Shin, and S. Lim, "Graph anomaly detection with graph neural networks: Current status and challenges," *IEEE Access*, 2022.

[14] K. Ding, J. Li, R. Bhanushali, and H. Liu, *Deep Anomaly Detection on Attributed Networks*, 05 2019, pp. 594–602.

[15] 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, pp. 12 07312 085, 2021.

[16] F. Zhang, H. Fan, R. Wang, Z. Li, and T. Liang, "Deep dual support vector data description for anomaly detection on attributed networks," *International Journal of Intelligent Systems*, vol. 37, no. 2, pp. 15091528, 2022.

[17] D. M. Tax and R. P. Duin, "Support vector data description," *Machine learning*, vol. 54, pp. 4566, 2004.

[18] X. Luo, J. Wu, A. Beheshti, J. Yang, X. Zhang, Y. Wang, and S. Xue, "Comga: Community-aware attributed graph anomaly detection," in *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, ser. WSDM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 657–665. [Online]. Available: https://doi.org/10.1145/3488560.3498389

[19] X. Yuan, N. Zhou, S. Yu, H. Huang, Z. Chen, and F. Xia, "Higher-order structure based anomaly detection on attributed networks," in *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 2021, pp. 26912700.

[20] Z. Peng, M. Luo, J. Li, L. Xue, and Q. Zheng, "A deep multi-view framework for anomaly detection on attributed networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 6, pp. 25392552, 2020.

[21] L.-H. Chen, H. Li, W. Zhang, J. Huang, X. Ma, J. Cui, N. Li, and J. Yoo, "Anomman: Detect anomalies on multi-view attributed networks," *Information Sciences*, vol. 628, pp. 121, 2023.

[22] K. Ding, J. Li, and H. Liu, "Interactive anomaly detection on attributed networks," in *Proceedings of the twelfth ACM international conference on web search and data mining*, 2019, pp. 357365.

[23] X. Song, M. Wu, C. Jermaine, and S. Ranka, "Conditional anomaly detection," *IEEE Transactions on knowledge and Data Engineering*, vol. 19, no. 5, pp. 631645, 2007.

[24] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[25] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, *et al.*, "Graph attention networks," *stat*, vol. 1050, no. 20, pp. 1048 550, 2017.

[26] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[27] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *The semantic web: 15th international conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, proceedings 15*. Springer, 2018, pp. 593607.

[28] X. Huang, Q. Song, J. Li, and X. Hu, "Exploring expert cognition for attributed network embedding," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 2018, pp. 270278.

[29] J. Li, X. Hu, J. Tang, and H. Liu, "Unsupervised streaming feature selection in social media," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 2015, pp. 10411050.

[30] X. Huang, Y. Yang, Y. Wang, C. Wang, Z. Zhang, J. Xu, L. Chen, and M. Vazirgiannis, "Dgraph: A large-scale financial dataset for graph anomaly detection," *Advances in Neural Information Processing Systems*, vol. 35, pp. 22 76522 777, 2022.

[31] S. García, J. Luengo, and F. Herrera, *Data preprocessing in data mining*. Springer, 2015, vol. 72.

[32] K. Ding, J. Li, R. Bhanushali, and H. Liu, "Deep anomaly detection on attributed networks," in *SIAM International Conference on Data Mining (SDM)*, 2019.

[33] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261272, 2020.

[34] X. Zheng, Y. Wang, Y. Liu, M. Li, M. Zhang, D. Jin, P. S. Yu, and S. Pan, "Graph neural networks for graphs with heterophily: A survey," 2024.

[35] X. Liu, M. Yan, L. Deng, G. Li, X. Ye, and D. Fan, "Sampling methods for efficient training of graph convolutional networks: A survey," *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 2, pp. 205234, 2021.

[36] T. Yu and H. Zhu, "Hyper-parameter optimization: A review of algorithms and applications," *arXiv preprint arXiv:2003.05689*, 2020.

[37] J. M. Thomas, A. Moallemy-Oureh, S. Beddar-Wiesing, and C. Holzhüter, "Graph neural networks designed for different graph types: A survey," *arXiv preprint arXiv:2204.03080*, 2022.