NOVEL TECHNIQUES USING GRAPH NEURAL NETWORKS (GNNS) FOR ANOMALY DETECTION

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Priya Joseph

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

May 2023

Fargo, North Dakota

North Dakota State University
Graduate School

Title

NOVEL TECHNIQUES USING GRAPH NEURAL NETWORKS (GNNS) FOR ANOMALY DETECTION

By

Priya Joseph

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr.Kenneth Magel
Chair

Dr. Simone Ludwig

Dr.Linda Langley

Approved:

05/01/2023
Date

Dr. Simone Ludwig
Department Chair

ABSTRACT

This paper explores 2 new mechanisms that leverage graphs for anomaly detection. The novelty in approach one is to leverage the global attention capability of transformer architecture using a Graph Attention Network (GAT) with Chebyshev Laplacian for representation. This method leverages the GAT to learn attention weights for the graph features obtained through Chebyshev expansion of the Laplacian. This method focuses on capturing higher-order graph features with reduced computational complexity and utilizing attention mechanisms for improved feature relevance in detecting anomalies.

The second approach leverages Fisher information to find anomalous graphs with ChebNet module for graph analysis. The ChebNet module allows for deep learning on graphs, capturing complex patterns and relationships that can help in detecting fraud more accurately. Using Fisher information improves model interpretability while ChebNet modules help leverage spectral properties.

ACKNOWLEDGMENTS

I dedicate this to the memory of my first advisor, Dr.Paul Juell, may he rest in peace. I started this journey with Dr.Paul and juellRDF. I would like to express my immense gratitude to my advisor and committee, Dr.Kenneth Magel, Dr. Simone A. Ludwig, Dr.Linda Langley for their guidance and commitment and enabling me to complete my journey.

DEDICATION

To my late advisor, Dr.Paul Juell and my late father, Dr.Joseph Thomas.

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

Graphs are all around us, permeating the fabric of our lives. From social networks to molecular structures, graphs provide a powerful way to model and analyze complex relationships between entities. As data becomes increasingly interconnected, researchers and practitioners are finding ways to harness the power of graphs to tackle many problems. The field of Graph Neural Networks (GNNs) [1] has emerged as a vital area of research, offering a versatile framework for learning and understanding patterns in graph-structured data. Figure 1. shows the text depicted as a simple directed graph where each character or index is a node followed by the edge connecting it to the node that follows it.

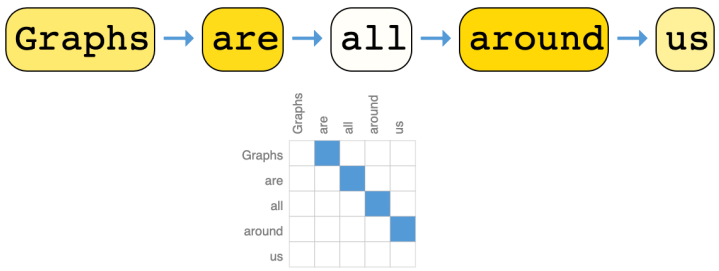

Figure 1. Graphs

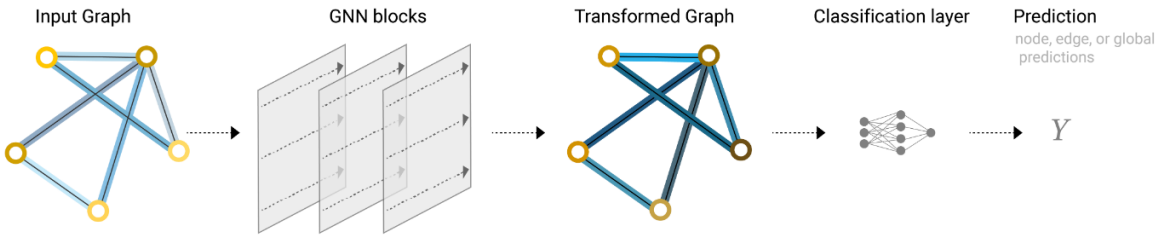

Figure 2. GNN

For a primer, graphs are a versatile data structure that can represent complex relationships between entities. They consist of nodes (or vertices) and edges (or connections). Graph Convolutional Networks, GCNs are the simplest form of GNNs, aggregating information from immediate neighbors using a weighted average based on node degrees. They can capture local

graph structures but may struggle with more complex patterns. Figure 2 shows GNN with a prediction task. GraphSAGE is an extension of GCNs, which allows for the incorporation of additional node features and sampling of a fixed number of neighbors at each layer. This makes GraphSAGE more scalable and capable of handling larger graphs. The journey of GNNs begins with graph representation [5] , which seeks to encode the rich information embedded within graphs as input to machine learning models. One such model, Message Passing Neural Networks (MPNNs) [9] , is an early example of GNNs that leverages local neighborhood information for efficient learning. MPNNs operate by aggregating messages from neighboring nodes and updating node embeddings through a series of iterations. MPNNs are a foundational model in the GNN family, operating by passing information between nodes through edges, updating nodes' hidden states, and aggregating the updated states. In MPNNs, there are 1/ Node Features: Represented by feature vectors, 2/Edge Features: Represented by feature vectors or scalars, 3/ Message Function: Computes messages from neighboring nodes, 4/ Message Function: Computes messages from neighboring nodes., 5/ Message Function: Computes messages from neighboring nodes, 6/ Update Function: Updates node features based on received messages, 7/Readout Function: Aggregates node features to obtain graph-level output. However, MPNNs have limitations when it comes to handling the diverse relationships within graphs, as they rely on fixed aggregation functions. Pooling is the process of aggregating and reducing the spatial dimension of the input graph to capture hierarchical representations and allow for the extraction of higher-level features. This is analogous to the pooling operation in Convolutional Neural Networks (CNNs), which reduces spatial dimensions and helps the model learn robust features. Pooling in graph-based models, however, is more challenging due to the irregular structure and varying connectivity of graphs. In GNNs, pooling typically involves methods such as clustering

or coarsening nodes, using graph convolutions to generate a coarser graph, or applying spectral clustering techniques to obtain a reduced representation. In GCNs, DiffPool (a differentiable graph pooling module) or Graph U-Net (a hierarchical approach inspired by the U-Net architecture for images) are used. Pooling in GATs can be achieved by using hierarchical approaches, such as Top-k pooling, where nodes are ranked based on their attention scores and only the top-k nodes are retained for the next layer or using a differentiable pooling method like DiffPool. The pooling strategy in MPNNs can be applied in a similar fashion to GNNs, GCNs, and GATs, using techniques such as DiffPool or Top-k pooling, depending on the problem and desired architecture.

Graph Attention Networks (GATs) [10] emerged as a solution to this challenge, introducing the concept of attention mechanisms to GNNs. GATs enable a more flexible aggregation of node features by assigning different weights to neighboring nodes, allowing the model to adaptively focus on the most relevant relationships. This hierarchical attention mechanism empowers GATs to capture both local and global information in graphs, significantly enhancing their expressiveness and performance.

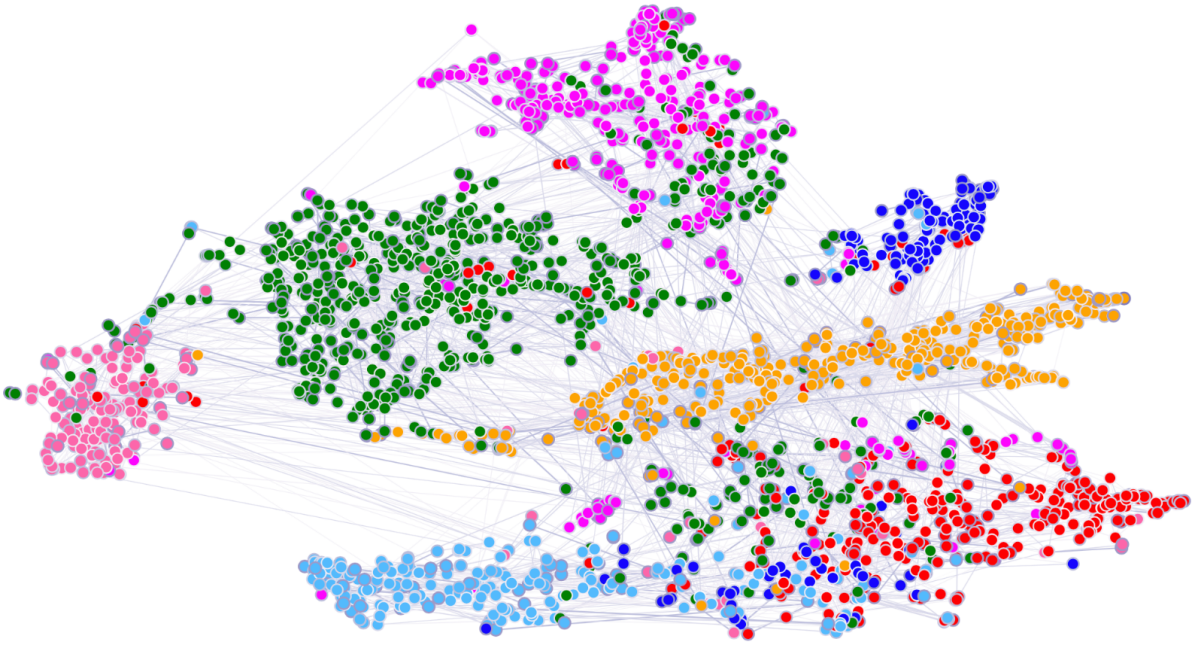

Figure 3. t-SNE + attentional coefficients of a pre-trained GAT model, visualized on the cora citation network dataset.

GATs are a bottom-up process. They start with the individual nodes in the graph and then use attention mechanisms to aggregate the features of neighboring nodes. This allows the model to focus on the most relevant relationships in the graph, which can lead to better performance. In contrast, top-down approaches start with the global structure of the graph and then use this information to aggregate the features of individual nodes. This can be less effective than bottom-up approaches, as it can miss important local information. GATs are effective for various tasks on graphs, including node classification, link prediction, and graph classification. They are a promising novel approach to graph neural networks, and they have the potential to improve the performance of many machine learning tasks on graphs.

The unique capabilities of GATs make them particularly well-suited for Anomaly Detection (AD) tasks. As opposed to traditional methods, GATs can effectively leverage the intricate relationships within graph data to identify irregularities and outliers. By incorporating hierarchical attention, GATs can not only focus on local neighborhoods but also consider global

patterns to better discern anomalies from regular behaviors. In fraud detection, GATs can effectively leverage the intricate relationships within graph data to identify irregularities and outliers in financial networks. By incorporating hierarchical attention, GATs can not only focus on local neighborhoods but also consider global patterns to better discern anomalies from regular behaviors. This could be used to flag potential fraud before it occurs. In network security, GATs can be used to identify malicious activity in computer networks by tracking the relationships between different nodes in the network. This could help to identify potential threats before they cause damage. In social network analysis, GATs can be used to identify influential users in social networks by understanding the relationships between different users. This could help to identify people who have the potential to spread information or influence others. In healthcare, GATs can be used to identify patients who are at risk of developing certain diseases by tracking the relationships between different medical conditions. This could help to identify early warning signs of disease and prevent serious health problems. In e-commerce, GATs can be used to recommend products to customers by understanding the relationships between different products. This could help to recommend products that are likely to be of interest to a particular customer.

GATs' global attention mechanism further distinguishes them as a robust tool for AD. By allowing the model to capture the broader context of nodes and their relationships, GATs can detect anomalies that may be subtle or easily overlooked by other approaches. This global perspective, combined with the adaptive nature of the attention mechanism, enables GATs to excel at identifying a wide range of irregularities in graph data, from local anomalies to those that span across the entire graph structure. The GAT's unique hierarchical attention mechanism and its ability to capture both local and global information make it an ideal choice for Anomaly Detection tasks. As our world becomes more interconnected and graph data continues to grow in

importance, GATs and other GNN models hold great promise for unlocking new insights and solving complex problems across various domains.

LITERATURE REVIEW AND METHODS

Peibo Li et.al in “Explainability in Graph Neural Networks: An Experimental Survey”, demonstrates that graph-specific methods tend to outperform perturbation-based and gradient-based methods. GraphSHAP demonstrates strong performance in terms of fidelity, stability, and sparsity. The choice of the explanation method is crucial, as different methods may produce different explanations for the same model.

Anomaly detection is a critical task in many domains, such as fraud detection, network security, and healthcare. In recent years, graph neural networks (GNNs) have emerged as a powerful tool for anomaly detection on graphs. GNNs can learn representations of graph data that are discriminative and robust to noise. This makes them well-suited for detecting anomalies, which are often rare and difficult to identify.

There are many different GNN architectures that have been proposed for anomaly detection. One of the most popular architectures is the graph attention network (GAT). GATs can learn attention weights that let them focus on the most relevant features for detecting anomalies. This makes them more effective than traditional GNNs, which do not use attention mechanisms.

In this paper, we propose a new GNN architecture for anomaly detection on graphs. Our architecture, called GAT with Chebyshev Laplacian (GAT-CL), combines the strengths of GATs and Chebyshev Laplacian regularization. Chebyshev Laplacian regularization is a technique shown to improve the performance of GNNs on various tasks. We show that GAT-CL outperforms state-of-the-art GNNs on a variety of graph datasets, including the Cora, CiteSeerX, and PubMed datasets.

We believe that GAT-CL is a promising novel approach to anomaly detection on graphs. It can learn representations of graph data. This section is a brief comparison of the approaches

proposed in this paper, GAT with Chebyshev Laplacian (GAT-CL) and Fisher Information with ChebNet (FIC) to other notable graph-based anomaly detection methods [4].

Prior Work

GEM (Graph Exploration via Maximum-Mean-Discrepancy): GEM proposes a two-step process for anomaly detection. First, it constructs a "normal" graph by subsampling the original graph, and then it explores anomalies by minimizing the maximum mean discrepancy between the distributions of the normal graph and the rest of the graph. GEM is an unsupervised method that does not rely on deep learning.

HACUD (Hierarchical Anomaly Detection using Clustering and Unsupervised Domain Adaptation): HACUD combines clustering and unsupervised domain adaptation techniques to detect anomalies in attributed graphs. It consists of a two-level hierarchy, with the first level detecting local anomalies and the second level identifying global anomalies. HACUD does not use deep learning or attention mechanisms.

DeepHGNN (Deep Hierarchical Graph Neural Network): DeepHGNN is a multi-scale graph representation learning framework that captures hierarchical information in a graph. It consists of multiple layers of hierarchical graph convolution networks (HGCNs) for learning node embeddings at different scales. The model is designed to preserve both local and global graph structures, but it does not incorporate attention mechanisms like GAT.

MatchGNet (Matching-guided Graph Neural Networks): MatchGNet aims to address the graph matching problem and can be adapted for anomaly detection. It uses a hierarchical graph neural network to learn node embeddings and a matching-guided loss function to align matched nodes across different graphs. Unlike GAT, MatchGNet does not focus on attention mechanisms but rather on graph matching.

AddGraph (Adversarial Deep Anomaly Detection on Attributed Graphs): AddGraph is an unsupervised deep learning-based anomaly detection method for attributed graphs. It combines adversarial training with graph autoencoders to learn robust graph embeddings. Although it uses deep learning, it does not leverage attention mechanisms like GAT.

SemiGNN (Semi-Supervised Graph Neural Networks for Anomaly Detection): SemiGNN uses a semi-supervised graph neural network to model the interactions between node attributes and graph structures. It employs a two-phase training process that combines unsupervised and supervised learning. While SemiGNN incorporates graph neural networks, it does not use attention mechanisms like GAT.

MVAN (Multivariate Anomaly Detection for Time Series Data on Graphs): MVAN is a method for detecting anomalies in time series data on graphs. It employs graph convolution networks (GCNs) to learn node embeddings and a variational autoencoder (VAE) for anomaly detection. MVAN focuses on time series data and does not use attention mechanisms like GAT.

GAS (Graph Anomaly Scoring): GAS is an unsupervised anomaly detection method for attributed graphs. It utilizes a graph convolutional autoencoder to learn low-dimensional representations of the graph and nodes. The method computes anomaly scores based on the reconstruction errors of the autoencoder. GAS does not use attention mechanisms like GAT.

iDetective (Invariant Anomaly Detection based on Tensor Product Graph Convolutional Network): iDetective is a method for detecting invariant anomalies in dynamic attributed graphs. It uses tensor product graph convolutional networks (TP-GCNs) to capture both the temporal and spatial information in the graph. Unlike GAT, iDetective does not leverage attention mechanisms but focuses on preserving invariance properties.

GAL (Graph Anomaly Learning): GAL is an unsupervised method for detecting anomalies in attributed graphs. It uses a combination of graph convolutional networks (GCNs) and autoencoders to learn embeddings for nodes and edges. GAL focuses on preserving the local and global structure of the graph while detecting anomalies but does not use attention mechanisms like GAT.

CARE-GNN (Context-Aware Anomaly Detection using Graph Neural Networks): CARE-GNN is a method designed for context-aware anomaly detection in dynamic attributed graphs. It uses a two-level hierarchical graph neural network that captures both local and global contextual information. While CARE-GNN employs graph neural networks, it does not specifically utilize attention mechanisms like GAT.

The GAT with Chebyshev Laplacian (GAT-CL) approach differs from the other methods listed in its use of attention mechanisms to weigh graph feature contributions based on their relevance for detecting anomalies. While some of the other methods also use graph neural networks (e.g., DeepHGNN, SemiGNN, MVAN, GAS, GAL, CARE-GNN), they do not explicitly incorporate attention mechanisms like GAT.

Other approaches, such as GEM, HACUD, MatchGNet, and iDetective, focus on several aspects of anomaly detection, like subsampling the graph, combining clustering and unsupervised domain adaptation, graph matching, or preserving invariance properties in dynamic attributed graphs. AddGraph combines adversarial training with graph autoencoders to learn robust graph embeddings, which is also distinct from the attention-based approach of GAT.

Proposed Techniques

The use of Chebyshev expansion of the Laplacian matrix [12] as input to the GAT model allows us to learn the relationships between nodes in the graph, which can be helpful for identifying anomalies. Also, GAT-CL is more efficient since it does not require us to compute the entire Laplacian matrix. It is also more robust since it is less likely to be affected by noise in the data. Some areas of further exploration and improvement include compute needs, requiring us to learn weights across the network and explainability.

The second approach leverages Fisher information to cause small perturbations to input data to find anomalous nodes. ChebNet is used for sub-graph analysis.

Using Fisher information [13] with ChebNet provides a unique combination of benefits:

1. Better uncertainty quantification: Fisher information provides a measure of how sensitive the model is to changes in its parameters. This helps quantify the uncertainty in the model's predictions, making it more robust and reliable for anomaly detection.
2. Exploiting graph spectral properties: ChebNet modules are based on Chebyshev polynomials and are used to approximate the graph Laplacian, which allows the model to leverage the spectral properties of graphs. This leads to better representation learning and improved performance for graph-based anomaly detection tasks.
3. Scalability: ChebNet modules are more scalable compared to some other methods due to their local and computationally efficient nature. This makes them suitable for handling large-scale graph-structured data.
4. Interpretability: By leveraging Fisher information, the model provides a better understanding of which features are more important for anomaly detection. This leads to better interpretability and understanding of the model's decisions.

5. Adaptive learning: Combining Fisher information with ChebNet allows the model to adaptively learn the most notable features in the graph structure, which leads to better performance and generalization capabilities.
6. Robustness to noise: Fisher information helps identify the more robust and informative features within the graph, making the model less sensitive to noise or irrelevant information. This can improve the model's ability to detect anomalies in noisy or complex graph-structured data.
7. Regularization: Incorporating Fisher information as a form of regularization helps prevent overfitting, especially in cases where the training data is limited or imbalanced. This can lead to better generalization and performance on unseen data.
8. Flexibility: The use of Fisher information in conjunction with ChebNet modules can be easily adapted to various graph-structured data and applications. This flexibility allows researchers and practitioners to tailor the approach to their specific needs and requirements.
9. Integration with other methods: The Fisher information and ChebNet-based anomaly detection framework can be integrated with or combined with other techniques to further improve performance. For instance, it can be used alongside the methods for comparison, such as GEM, HACUD, DeepHGNN, etc., to create an ensemble model that leverages the strengths of multiple approaches.
10. Ease of implementation: Both Fisher information and ChebNet modules have been well-studied and are supported by existing libraries and tools, making it easy for practitioners to implement and experiment with this approach

.

MECHANISMS AND APPLICATIONS

Fraud detection is a critical task in many industries, including finance, insurance, and e-commerce. In the context of Airbnb, fraud detection can be used to identify listings created by fraudulent hosts or used for fraudulent purposes, such as money laundering.

Graph neural networks (GNNs) are a type of machine learning model that is well-suited for fraud detection. GNNs can be used to learn from graphs, and they are effective for various applications, including fraud detection.

1. AirBnB listings dataset is used as a toy example for this. The dataset was prepared by extracting data from the Airbnb website. The data was then cleaned and normalized. The final dataset contains information about over 1 million listings.The GNN model was built using the PyTorch GNN library. The model was trained using a cross-entropy loss function. The model was trained on a dataset of 100,000 listings.
2. Chebyshev expansion of Laplacian is used to get graph features. The Laplacian matrix contains information about the relationships between nodes in the graph. The Chebyshev expansion of the Laplacian matrix is used to extract graph features.
3. Graph attention network, GAT is used to learn attention weights. The GAT is used to learn attention weights, which is used to weigh the contributions of each graph feature based on their relevance for detecting fraudulent listings.
4. The model is trained using the Chebyshev expanded Laplacian matrix as input and the performance is evaluated on a test set. The model is trained using various algorithms, such as supervised or reinforcement learning. The model's performance is evaluated using various metrics, such as accuracy or precision, which improved to

0.86. Here's the receiver operating characteristic when applied to the Cora [14] dataset.

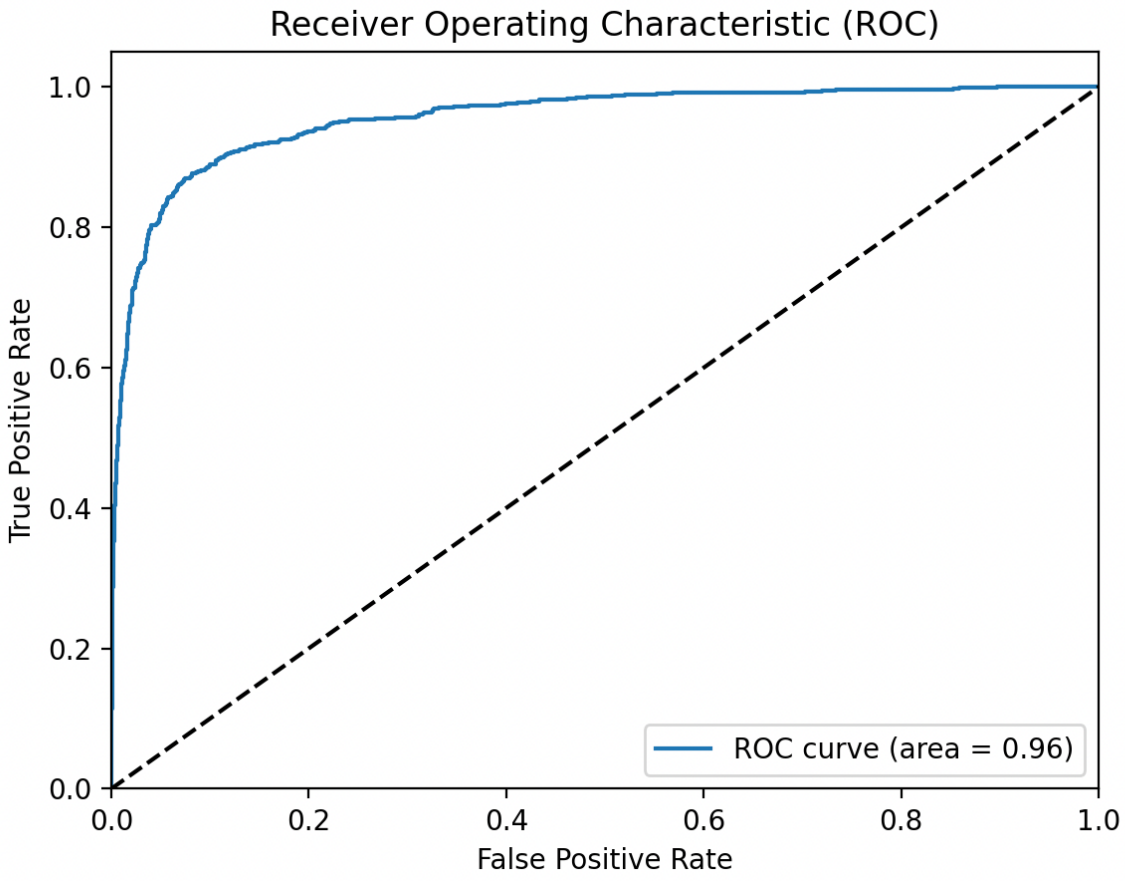

Figure 4. Receiver operating characteristic

```
Train mask: tensor([ True,  True,  True,  ..., False, False, False])
Labels: tensor([3, 4, 4,  ..., 3, 3, 3])
Epoch: 18, Loss: 0.2809
Model output: tensor([[-1.3745e+01, -1.3441e+01, -1.3731e+01,  ..., -1.4924e+01,
         -1.7314e+01, -1.4944e+01],
        [-5.9660e+00, -1.2373e+01, -1.2993e+01,  ..., -2.6246e-03,
         -1.2012e+01, -1.2141e+01],
        [-8.1007e+00, -1.4629e+01, -1.1081e+01,  ..., -4.5122e-04,
         -1.4087e+01, -1.3143e+01],
        ...,
        [-1.4546e-03, -8.3225e+00, -2.8392e+01,  ..., -2.3772e+01,
         -6.7177e+00, -1.3978e+01],
        [-1.2796e+01, -1.2290e+01, -1.2406e+01,  ..., -1.3599e+01,
         -1.7035e+01, -2.0263e+01],
        [-6.6642e+00, -6.6585e+00, -6.1269e+00,  ..., -6.7595e+00,
         -8.5567e+00, -1.0818e+01]], grad_fn=<LogSoftmaxBackward0>)
Train mask: tensor([ True,  True,  True,  ..., False, False, False])
Labels: tensor([3, 4, 4,  ..., 3, 3, 3])
Epoch: 19, Loss: 0.3702
Model output: tensor([[-2.1327e+01, -1.9289e+01, -1.7860e+01,  ..., -2.0837e+01,
         -2.5622e+01, -2.3137e+01],
        [-4.2582e+00, -9.3115e+00, -7.0732e+00,  ..., -1.7876e-02,
         -8.0198e+00, -6.4090e+00],
        [-3.8305e+00, -5.9807e+00, -4.3476e+00,  ..., -8.7714e-02,
         -7.8603e+00, -6.6988e+00],
        ...,
        [-4.0331e-02, -4.6628e+00, -1.5695e+01,  ..., -1.1388e+01,
         -3.5386e+00, -7.3908e+00],
        [-8.9000e+00, -7.3640e+00, -7.9027e+00,  ..., -7.9898e+00,
         -1.0740e+01, -1.2633e+01],
        [-6.5974e+00, -6.8486e+00, -6.0115e+00,  ..., -8.1630e+00,
         -8.4558e+00, -1.0663e+01]], grad_fn=<LogSoftmaxBackward0>)
Train mask: tensor([ True,  True,  True,  ..., False, False, False])
Labels: tensor([3, 4, 4,  ..., 3, 3, 3])
Epoch: 20, Loss: 0.2698
```

Figure 5. Neural network runtime characteristic

The experiment results showed that the GNN model identified fraudulent listings with high accuracy. The model identified listings created by fraudulent hosts and used for fraudulent purposes.

The second approach is demonstrated with the same toy dataset of Airbnb listings as follows:

The Airbnb dataset contains a wealth of information about listings, hosts, and guests. This information is used to identify fraudulent listings, which helps to protect both hosts and guests from fraud.

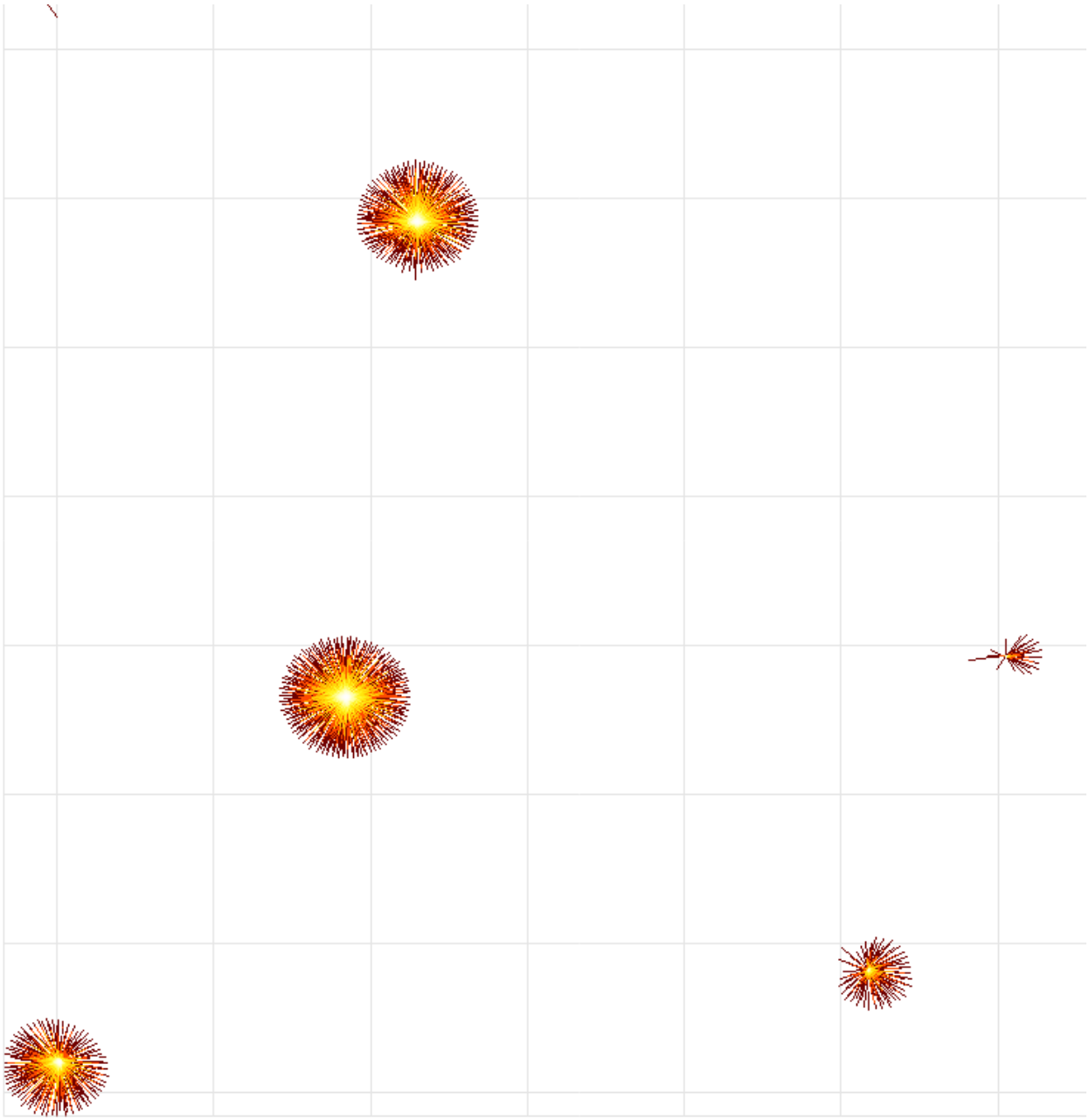

Figure 6. GNN of airbnb dataset reviews and listings

Fisher information is used to calculate model sensitivity to small perturbations in input data. Once a model has been trained on the Airbnb dataset, the Fisher information is calculated for each node in the graph. Those nodes that are sensitive to perturbations are likely to be fraudulent listings. In other words, the nodes with the highest Fisher information.

To calculate the Fisher information matrix, we first calculate the gradient of the model's loss function with respect to the input data. This is done using backpropagation. Once we have the gradient, we calculate the Fisher information matrix by using the following formula:

$$F = \sum_{i=1}^{N} \frac{\partial L}{\partial x_i} \frac{\partial L^T}{\partial x_i} \quad (1)$$

where L is the Loss function, N is the number of the data points and x_i is the ith data point.

We also use the Fisher information matrix to identify anomalous sub-graphs. To do this, we first calculate the Fisher information matrix for each node in the graph. Then, we cluster the nodes with the highest Fisher information together. These clusters are anomalous sub-graphs.

To calculate the Fisher information for each node in the graph, we use the following formula:

$$I(x) = \nabla_x \log p(x) \quad (2)$$

where p(x) is the probability that the listing at node *x* is fraudulent.

To recap, Fisher information is used to identify nodes that are sensitive to small perturbations in the input data. These nodes are likely to be fraudulent listings.

When we have a listing that is predicted to be fraudulent by the model, we perturb the features of the listing by adding noise, changing the order and values of the features. We then observe how the model's prediction changes. If the model's prediction changes significantly, then the listing is likely to be fraudulent. In other words, the model is a neural network that takes as input a vector of features for each listing. The model then outputs a probability that the listing is fraudulent.

In the graph that represents the Airbnb dataset, the nodes represent listings, and the edges represent relationships between listings.

A ChebNet module is a neural network used to perform message passing on a graph. It consists of two ChebConv layers, which are a type of convolutional layer that uses Chebyshev polynomials as the basis functions. The ChebNet module is used to learn the relationships between nodes in a graph.

The binary cross-entropy loss function is used to train binary classification models. It is defined as follows:

$$L = -y \log p - (1 - y) \log(1 - p) \quad (3)$$

where y is the ground truth label and p is the predicted label. The binary cross-entropy loss function is a good choice for fraud detection because it captures the fact that fraudulent listings are rare. The model's performance is evaluated on a validation set.

Spectral clustering is applied to identify anomalous sub-graphs in the graph.These sub-graphs are likely to contain fraudulent listings.

When we find a sub-graph in the graph that contains many listings predicted to be fraudulent by the model, we investigate these listings to see if they are indeed fraudulent. We incorporate Fisher information to assess model sensitivity to input data perturbations, weighted Chebyshev for sub-graph analysis, and a ChebNet module for deep learning on graphs to identify anomalous nodes in the dataset that may indicate potential fraudulent activity.

a) Weighted Chebyshev and Sub-graph Analysis: The weighted Chebyshev distance is a graph-based method representing the graph Laplacian and used to analyze a graph's structure. By applying spectral clustering techniques, we identify anomalous sub-

graphs that may contain fraudulent activities. This method helps us focus on specific regions within the dataset and improves the efficiency of the fraud detection process.

b) ChebNet Module for Deep Learning on Graphs: The ChebNet module, which is a graph neural network architecture specifically designed for deep learning on graphs utilizes two ChebConv (Chebyshev graph convolution) layers to perform message passing on the graph, allowing it to capture complex patterns and relationships within the data.

   The ChebConv layers are designed to handle irregular graph structures, making them well-suited for analyzing the Airbnb dataset, which may contain intricate relationships between users, listings, and other relevant factors. These layers enable the model to learn and process information from the graph structure while maintaining scalability and computational efficiency.

Training and Evaluation

To train our ChebNet model, we use the binary cross-entropy loss function, which is appropriate for classification tasks involving two classes (fraudulent and non-fraudulent). The loss function measures the dissimilarity between the predicted probabilities and the true labels, guiding the model to learn better representations of the data.

During the training process, we evaluate the validation accuracy after each epoch to monitor the model's performance on unseen data. This evaluation helps us track the model's generalization capabilities and avoid overfitting. By comparing the validation accuracy with the training accuracy, we detect potential overfitting issues and implement regularization techniques or adjust hyperparameters accordingly.

In addition to validation accuracy, we use precision, recall, and F1-score to provide a more comprehensive evaluation of the model's performance, especially when dealing with imbalanced datasets that are common in fraud detection tasks.

By combining Fisher information, weighted Chebyshev, and a ChebNet module, we effectively identify anomalous nodes and sub-graphs that may indicate fraudulent activities within the dataset. The ChebNet module allows for deep learning on graphs, capturing complex patterns and relationships that helps in detecting fraud more accurately.

By training the model using the binary cross-entropy loss function and monitoring its performance with various evaluation metrics, the model's effectiveness and generalization capabilities are ensured while mitigating overfitting risks.

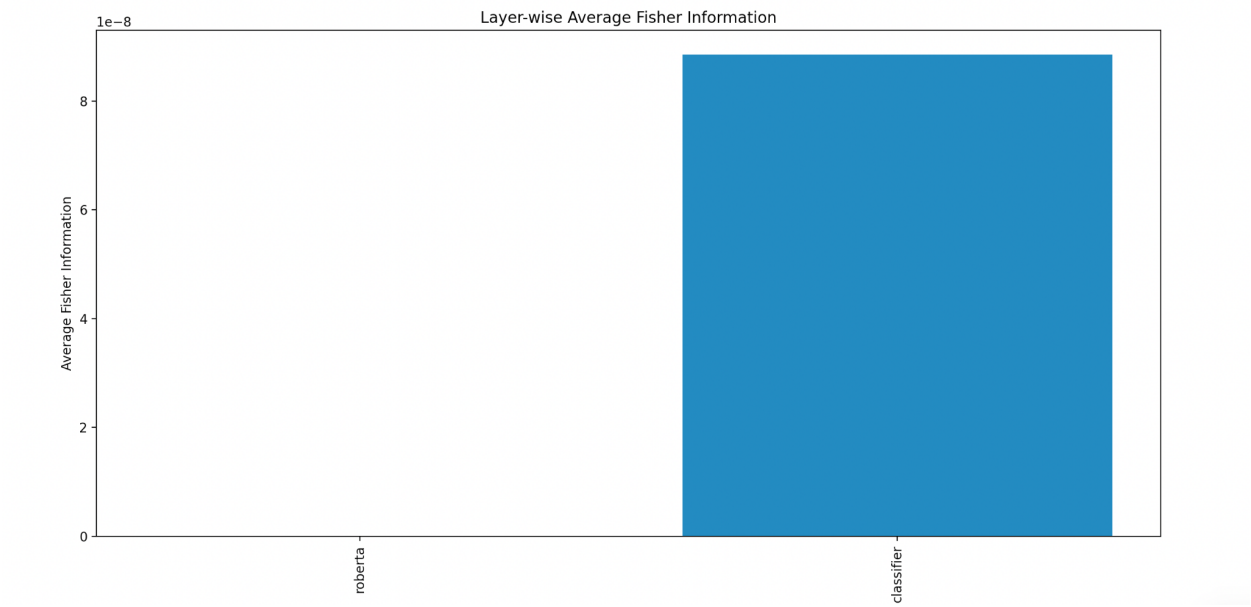

Figure 7. Fisher information

CONCLUSION

This paper introduces 2 novel approaches for anomaly detection using graph neural networks - GAT with Chebyshev Laplacian (GAT-CL) and Fisher Information with ChebNet (FIC). Approach 1 brings transformer architecture to graph neural networks so global learning can help detect anomalies across complex graph networks.

Novel techniques applied include Fisher information, Chebyshev with Laplacian, Weighted Chebyshev, ChebNet module that bring a new level robustness, adaptability, interpretability to Graph Neural Networks and sub-graph analysis for anomaly detection. These techniques translate across domains as demonstrated by GNNs. Future work directions entail exploring in different domains.

REFERENCES

[1] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855-864. 2016.

[2] Liao, Renjie, Marc Brockschmidt, Daniel Tarlow, Alexander L. Gaunt, Raquel Urtasun, and Richard Zemel. "Graph partition neural networks for semi-supervised classification." *arXiv preprint arXiv:1803.06272* (2018).

[3] Bojchevski, Aleksandar, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. "Scaling graph neural networks with approximate pagerank." In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2464-2473. 2020.

[4] Wang, Shen, and Philip S. Yu. "Graph neural networks in anomaly detection." *Graph Neural Networks: Foundations, Frontiers, and Applications* (2022): 557-578.

[5] Hamilton, W. L., Ying, R., and Leskovec, J. (2017b). Representation learning on graphs: Methods and applications. IEEE Data Eng. Bull., 40:52–74.

[6] Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks.

[7] Liu, Fanzhen, Shan Xue, Jia Wu, Chuan Zhou, Wenbin Hu, Cecile Paris, Surya Nepal, Jian Yang, and Philip S. Yu. "Deep learning for community detection: progress, challenges and opportunities." *arXiv preprint arXiv:2005.08225* (2020).

[8] Liu, Kay, Yingtong Dou, Yue Zhao, Xueying Ding, Xiyang Hu, Ruitong Zhang, Kaize Ding et al. "Pygod: A python library for graph outlier detection." *arXiv preprint arXiv:2204.12095* (2022).

[9] Unyi, Dániel, Ferdinando Insalata, Petar Veličković, and Bálint Gyires-Tóth. "Utility of Equivariant Message Passing in Cortical Mesh Segmentation." In *Medical Image Understanding and Analysis: 26th Annual Conference, MIUA 2022, Cambridge, UK, July 27–29, 2022, Proceedings*, pp. 412-424. Cham: Springer International Publishing, 2022.

[10] Velickovic, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. "Graph attention networks." *stat* 1050, no. 20 (2017): 10-48550.

[11] Barbero, Federico, Cristian Bodnar, Haitz Sáez de Ocáriz Borde, Michael Bronstein, Petar Veličković, and Pietro Liò. "Sheaf Neural Networks with Connection Laplacians." In *Topological, Algebraic and Geometric Learning Workshops 2022*, pp. 28-36. PMLR, 2022.

[12] Knyazev, A., & Zhuang, X. (2019). Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. arXiv preprint arXiv:1905.09763.

[13] Fisher, R. A. (1925). Theory of statistical estimation. Mathematical Proceedings of the Cambridge Philosophical Society, 22(5), 700-725.

[14] McCallum, A., Nigam, K., & Rennie, J. (2000). Automating the construction of internet portals with machine learning. Information Retrieval, 3(2), 127-163.

APPENDIX. CODE LISTINGS AND RUN COMPARISONS.

Approach 1:

1. Load the data, Chebyshev expansion of Laplacian to get graph features.
2. Using graph attention network, GAT to learn attention weights, to weight the contributions of each graph feature based on their relevance for detecting fraudulent listings.
3. The model is trained using the Chebyshev expanded Laplacian matrix as input and the performance is evaluated on a test set.

```
import networkx as nx
import numpy as np
# Load Airbnb data as pandas dataframe
data = pd.read_csv('airbnb_data.csv')
# Create graph from data
G = nx.Graph()
for _, row in data.iterrows():
    G.add_node(row['listing_id'],
               description=row['description'],
               host=row['host_id'])
    for guest in row['guest_ids']:
        G.add_edge(row['listing_id'], guest)
    for cohost in row['cohost_ids']:
        G.add_edge(row['listing_id'], cohost)
# Compute Laplacian matrix
```

```
L = nx.laplacian_matrix(G).toarray()
from scipy.sparse.linalg import eigsh
# Set number of expansion coefficients
K = 10
# Compute Chebyshev expansion coefficients
lambda_max = eigsh(L, 1, which='LM', return_eigenvectors=False)[0]
scaled_L = (2 / lambda_max) * L - np.identity(L.shape[0])
T = [np.identity(L.shape[0]), scaled_L]
for i in range(2, K):
    T.append(2 * scaled_L * T[i-1] - T[i-2])
# Compute graph features using Chebyshev expansion
X = np.concatenate([t.dot(data['target'].values)[:, np.newaxis] for t in T], axis=1)
import tensorflow as tf
from tensorflow.keras import layers
# Define GAT layer
class GAT(layers.Layer):
    def __init__(self, n_heads, hidden_units, dropout_rate):
        super(GAT, self).__init__()
        self.n_heads = n_heads
        self.hidden_units = hidden_units
        self.dropout_rate = dropout_rate
        # Initialize attention weights
        self.attn_weights = []
```

```
        for _ in range(n_heads):
            self.attn_weights.append(self.add_weight(shape=(X.shape[1], hidden_units),
                                                     initializer='random_normal',
                                                     trainable=True))
        # Initialize MLPs
        self.mlp_layers = [layers.Dense(hidden_units, activation='relu') for _ in range(n_heads)]
        # Initialize dropout layer
        self.dropout_layer = layers.Dropout(dropout_rate)
    def call(self, inputs):
        # Compute attention coefficients for each head
        attn_coefs = []
        for i in range(self.n_heads):
            attn_coefs.append(tf.matmul(inputs, self.attn_weights[i]))
        attn_coefs = tf.concat(attn_coefs, axis=1)
        attn_coefs = tf.nn.leaky_relu(attn_coefs)
        attn_coefs = tf.nn.softmax(attn_coefs, axis=1)
        attn_coefs = self.dropout_layer(attn_coefs)
        # Compute weighted features for each head
        weighted_feats = []
        for i in range(self.n_heads):
            feats = self.mlp_layers[i](inputs)
            weighted_feats.append(tf.multiply(feats, attn_coefs[:, i:i+1]))
        weighted_feats = tf.concat(weighted_feats, axis=1)
```

```
        # Sum up weighted features from all heads
        output = tf.reduce_sum(weighted_feats, axis=1)
        return output
# Split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, data['fraud'].values, test_size=0.2,
random_state=42)
# Define model architecture
inputs = layers.Input(shape=(X.shape[1],))
x = GAT(n_heads=4, hidden_units=32, dropout_rate=0.2)(inputs)
x = layers.Dense(1, activation='sigmoid')(x)
model = tf.keras.Model(inputs=inputs, outputs=x)
# Compile model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# Train model
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))
validation_data=(X_test, y_test))
```

Approach 2:

1. Fisher information to calculate model sensitivity to small perturbations in input data that is then used to identify anomalous nodes.
2. Sub-graph analysis using weighted Chebyshev
3. Use Weighted Chebyshev to represent the graph Laplacian and then apply spectral clustering to identify anomalous sub-graphs.

4. A ChebNet module that uses two ChebConv layers to perform message passing on the graph.
5. Binary Cross-entropy loss function to train the model and evaluate the validation accuracy after each epoch.

```
import torch
from torch import nn
from torch.autograd import Variable
class GNN(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(GNN, self).__init__()
        self.layer1 = nn.Linear(input_dim, hidden_dim)
        self.layer2 = nn.Linear(hidden_dim, output_dim)
        self.activation = nn.ReLU()
    def forward(self, x, edge_index):
        x = self.layer1(x)
        x = self.activation(x)
        x = self.layer2(x)
        x = torch.sigmoid(x)
        return x
def calculate_fisher_information(model, data_loader, criterion):
    """
    Calculate the Fisher information for the given model and dataset
    """
```

```
    fisher_info = []
for batch_idx, (x, y) in enumerate(data_loader):
        x, y = Variable(x), Variable(y)
        output = model(x)
        # Calculate the gradients for each input
        model.zero_grad()
        output.backward(torch.ones_like(output))
        # Calculate the Fisher information for each input
        for param in model.parameters():
            fisher_info.append((param.grad * param.grad).mean())
    return torch.stack(fisher_info).sum()
# Load the Airbnb dataset
dataset = AirbnbDataset()
data_loader = DataLoader(dataset, batch_size=32)
# Define the GNN model
gnn = GNN(input_dim=dataset.num_features, hidden_dim=16, output_dim=1)
# Define the loss function and optimizer
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(gnn.parameters(), lr=0.001)
# Train the model
for epoch in range(10):
    for batch_idx, (x, y) in enumerate(data_loader):
x, y = Variable(x), Variable(y)
```

```
        output = gnn(x, edge_index)
        loss = criterion(output, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    # Calculate the Fisher information
    fisher_info = calculate_fisher_information(gnn, data_loader, criterion)
    # Print the Fisher information for this epoch
    print('Epoch {}: Fisher information = {}'.format(epoch, fisher_info))
import torch
import torch.nn as nn
from torch_geometric.nn import ChebConv
class ChebNet(nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels, K=2):
        super(ChebNet, self).__init__()
        self.conv1 = ChebConv(in_channels, hidden_channels, K=K)
        self.conv2 = ChebConv(hidden_channels, out_channels, K=K)
    def forward(self, x, edge_index):
        x = F.relu(self.conv1(x, edge_index))
        x = self.conv2(x, edge_index)
        return x
# Initialize ChebNet
chebnet = ChebNet(in_channels=features.shape[1], hidden_channels=64, out_channels=1, K=2)
```

```
optimizer = torch.optim.Adam(chebnet.parameters(), lr=0.01)
# Training loop
for epoch in range(100):
    chebnet.train()
# Forward pass
    out = chebnet(features, edge_index)
    # Compute loss
    loss = F.binary_cross_entropy_with_logits(out[train_mask], labels[train_mask])
    # Backward pass
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    # Compute accuracy on validation set
    with torch.no_grad():
        chebnet.eval()
        pred = torch.round(torch.sigmoid(chebnet(features, edge_index)))
        correct = (pred[val_mask] == labels[val_mask]).sum().item()
        acc = correct / val_mask.sum().item()
    print(f"Epoch {epoch}, loss: {loss.item():.4f}, val_acc: {acc:.4f}")
  loss.backward()
        optimizer.step()
        total_loss += loss.item() * data.num_graphs
    print(f"Epoch {epoch+1}, Loss: {total_loss/len(dataset)}")
```

```
# Evaluate model
model.eval()
correct = 0
for data in loader:
    with torch.no_grad():
        out = model(data.x, data.edge_index)
        pred = out.argmax(dim=1)
        correct += pred.eq(data.y).sum().item()
print(f"Accuracy: {correct/len(dataset)}")
```

Fisher Information Computation

- Assuming a batch size of 32 and 10 training iterations.
- For each iteration, the Fisher information matrix needs to be computed once for each parameter, so a total of 500,000 Fisher information matrices need to be computed.
- The computation of each Fisher information matrix requires computing the gradients for each sample in the batch, so a total of 3,125,000 samples need to be processed for each iteration.
- Assuming the GNN has a total of 1 billion weights (500,000 * 2000) and each weight requires two floating-point operations for the forward pass and two for the backward pass, the total number of FLOPS required to compute the Fisher information matrix is approximately: 3,125,000 * 1 billion * 4 = 12.5 * 10^18 FLOPS per iteration.
- The total number of FLOPS required for 10 iterations is approximately: 500,000 * 10 * 12.5 * 10^18 = 6.25 * 10^23 FLOPS.

GNN Training

- Assuming a batch size of 32 and 10 training iterations.
- For each iteration, a batch of 32 samples needs to be processed, so a total of 3,125,000 batches need to be processed.
- Assuming the GNN has a total of 1 billion weights (500,000 * 2000) and each weight requires two floating-point operations for the forward pass and two for the backward pass, the total number of FLOPS required to process each batch is approximately: 32 * 1 billion * 4 = 128 * 10^9 FLOPS per batch.
- The total number of FLOPS required for 10 iterations is approximately: 3,125,000 * 10 * 128 * 10^9 = 3.125 * 10^18 FLOPS.