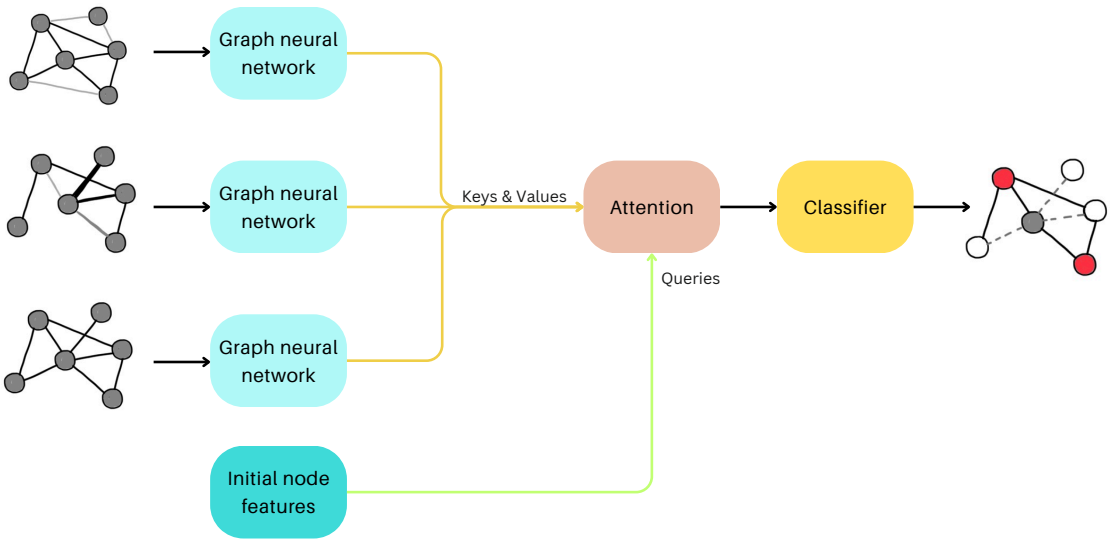
Self Attending Edges: Towards a Simpler Anomaly Detection on Graphs

Joan Lafuente, Neil De La Fuente, Maiol Sabater, and Daniel Vidal

Universitat Autònoma de Barcelona

June 2024



Contents

1 Introduction

1.1 Context and Motivation

In the digital landscape, the rise of cyber fraud, ranging from fake reviews to bank scams, poses significant risks to both consumers and businesses. Financial transactions, as well as interactions on the Internet, commonly structure complex networks of graph data that enhance the representation of these interactions. Detecting anomalies inside those graphs guarantees data integrity, builds trust, and protects economic interests.

Thus, accurate anomaly detection on graph-based systems ensures significant benefits within different domains. Digital service providers must identify and mitigate fraudulent behaviors against their users at all costs. In our case study with item review datasets, effective systems have proven to safeguard companies interests against financial losses, protect the brand's reputation, and ensure compliance with regulations. For users, robust anomaly detection adds further strength to the guarantee of online reviews and recommendations, allowing more informed buying decisions. Such transparency builds consumer confidence and creates a healthier online marketplace. Businesses benefit from better data quality, leading to more precise market insights and better decision-making processes. Improved data integrity promotes the development of much more reliable machine learning models, thus advancing technology and research.

1.2 Related Work

The landscape of graph anomaly detection has seen significant advancements through various state-of-the-art contributions. These approaches go from attribute-driven graph representations to spectral graph neural networks, each offering unique strengths and novel methodologies for dealing with the complexities inherent in graph-based anomaly detection.

Attribute-Driven Graph Representation One prominent work by Xiang et al. introduces a semi-supervised approach for credit card fraud detection utilizing attribute-driven graph representations. This method embeds various attributes such as card type, currency ID, and merchant type to create a comprehensive graph structure. The key innovation lies in the Gated Temporal Attention Network (GTAN), which facilitates message passing among nodes to learn transaction representations over time. This temporal graph attention mechanism effectively captures the dynamics of transactions, making it adept at identifying fraudulent activities.

Spectral Graph Neural Network (SplitGNN) Another notable contribution is SplitGNN by Xu et al., which addresses the challenge of heterophily in graphs—where linked nodes often have dissimilar features or labels. SplitGNN mitigates this by splitting graphs into heterophilic and homophilic edges, thereby capturing more significant expressions of signals across different frequency bands. This spectral approach enhances the model's ability to detect fraud by focusing on the distinct characteristics of connected nodes, ultimately improving overall detection accuracy.

1.3 Summary of The Proposed Solution

To assess the problem of detecting fraudulent activities, we treat such data as graph data, allowing for more effective processing and detection of anomalies within complex graph

structures. In addition to this, we propose a novel architecture that uses Graph Neural Networks (GNNs) to provide better representations for the activities. The detailed explanation can be found in Section 2.2.

Our approach involves treating different edge types as subgraphs and processing them individually through distinct GNNs. This method allows us to capture the unique characteristics and relationships inherent to each edge type more effectively.

After processing, we employ a self-attention mechanism to enable the different edge types to attend to each other, thereby extracting more meaningful relations. The node features derived from this process, along with the initial node features, are then passed through another self-attention layer. This comprehensive set of features is subsequently fed into the final classifier head to detect anomalies. A visualization of the pipeline/architecture is provided in 1.

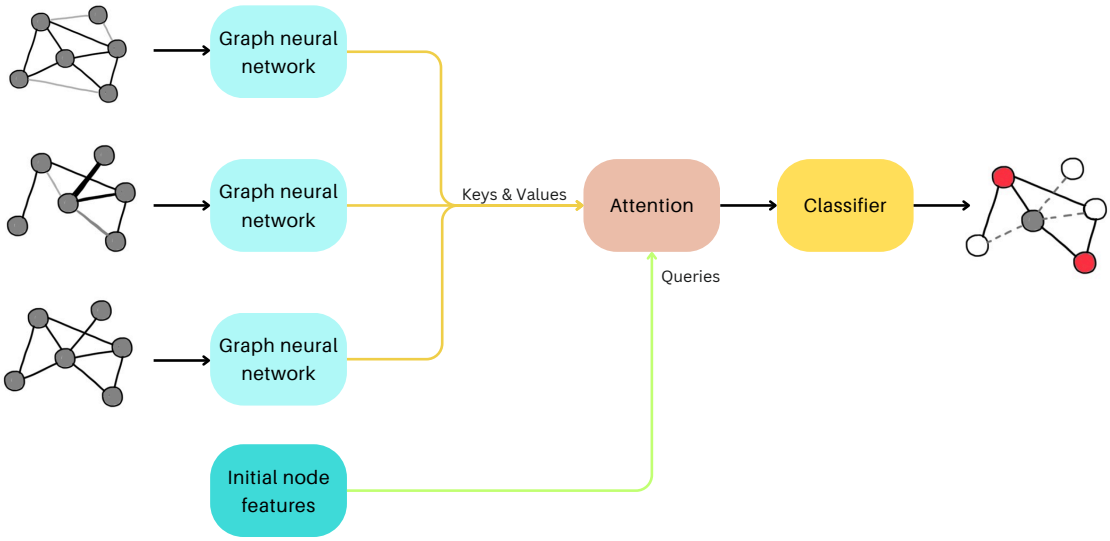


Figure 1: Proposed Architecture.

Novel Contributions Our proposed model advances the state-of-the-art by introducing a novel edge self-attention framework that treats different edge types as subgraphs. Unlike previous works that typically process all edge types uniformly, our approach leverages the strengths of various GNN architectures to capture diverse patterns and relationships in the graph data. This is achieved by:

1. **Processing Each Edge Type Individually:** Each type of edge is treated as a separate subgraph and processed through its specific GNN, allowing for a more granular and detailed representation of each subgraph.

2. **Self-Attention Mechanism:** After individual processing of each subgraph, a self-attention mechanism allows different node features extracted from the different edge types to interact and attend to each other, facilitating the extraction of more meaningful and complex relationships.

3. **Efficient and Scalable:** Our method demonstrates remarkable efficiency, achieving industry-level performance using only 50% of the training data. This data efficiency is a critical advancement, particularly for applications where labeled data is scarce.

Additionally, our model operates in real-time on standard CPU hardware, showcasing its scalability and practical applicability in real-world scenarios.

By addressing the computational challenges associated with handling multiple edge types and optimizing self-attention mechanisms, our approach offers a novel and comprehensive solution. These innovations position our model as a robust and versatile tool for detecting anomalies in complex graph-structured data across various domains, contributing to the development of scalable and efficient graph anomaly detection models.

2 Proposal of Solution

2.1 Objectives

This project aims to develop a powerful tool that can be used to detect anomalies from graph data achieving a reliable performance in real-time trying to depend on a low amount of resources and ensuring good scalability of our solution.

Also, we aim to use the less amount of labeled data as possible, as in this domain can be difficult to obtain it. Lastly, it is important that the developed tool can be adapted to work on other data or domains.

2.2 Artificial Intelligence Methodology

To address the problem, we propose the use of a base architecture that can be seen in Figure 1. In this architecture we use a different graph neural network for each of the types of connections that appear on the graph, we name backbone the base architecture used on each of them.

This separate processing of the graph information, instead of using a single graph neural network with the connection type as an edge feature, allows learning more insightful features of each of the graph structures present. In the graphs that we have used, we consider it very important as each of the connection types means very different facts, and processing it jointly does not allow us to learn additional information, moreover, it affects negatively.

To join this information learned from each type of connection we decided to use dot product attention, this allows us to combine the feature vectors taking into account the original node attributes. The keys and values needed for the attention mechanism, are extracted by applying batch normalization, a linear layer and an activation function to the features obtained from each of the graph neural networks. The queries are extracted from the original node features in the same way as keys and values. So, by using an attention mechanism we combine the feature vectors of each of the graph neural networks, three in our case, into a single feature vector that describes the node.

Once we have this single feature vector describing the node, we use a classification head in order to detect the anomalies. For this task, we have tried several options which are explained in section 4.2.

Lastly, the different training methods tried on this base architecture and the variations tried are explained in section 4, and the data used on these experiments is explained in section 3.

2.3 Project Planning

As shown in Figure 2, our timeline consists of seven distinct stages, each representing an evolution of the preceding one.

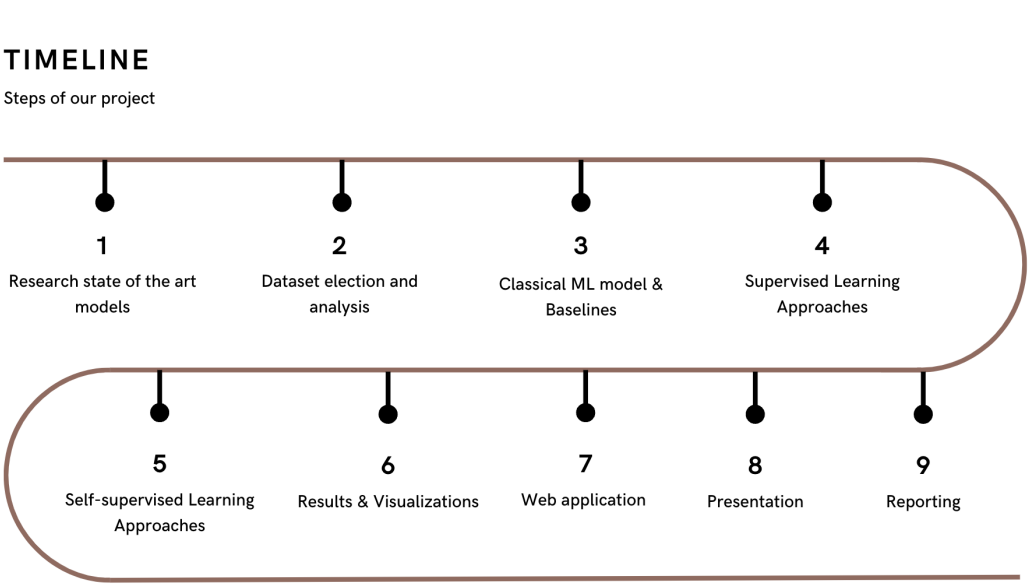


Figure 2: Timeline

1. **Research State-of-the-Art Models**: Initially, the team researched the latest models and methods for solving the problem of detecting anomalies in graphs.

2. **Dataset Selection and Analysis**: The Amazon dataset was initially chosen subsequently, the Yelp dataset was also selected due to the ease of Amazon's dataset to classify. Preliminary analyses were conducted during the early weeks.

3. **Classical Machine Learning Model & Baselines**: Several implementations were tested, achieving promising results due to pre-processed attributes.

4. **Supervised Learning Approaches**: After experimenting with ML models and observing their limited performance on the Yelp dataset compared to the Amazon dataset, both supervised and autoencoder approaches were explored.

5. **Results & Visualizations**: We extracted meaningful metrics for all models developed and compared them to evaluate their performance and did some plots to represent these results in an easy-to-understand way.

6. **Self-Supervised Learning**: We hypothesized that self-supervised approaches would yield better results, leading to the development of a self-supervised model.

7. **Web Application Development**: After evaluating different models and approaches, a web application was built to serve as a reliable product for companies, incorporating the best-performing deep learning models.

8. **Presentation**: We prepared a brief presentation showing all the work done during this project as well as the results and Demo which you can watch in the following link: *Self Attending Edges*.

9. **Reporting**: Finally, all results, procedures, and methods were documented comprehensively in the present report. There is also a part of code documentation that

consists of a GitHub repository where there is available all the code and documentation to reproduce our experiments and test the web application in the following link: *https://github.com/joanlafuente/SAE*.

In the following Figure 3, a simplified Gantt Chart illustrates the time we spent on each of the groups of tasks, when we started them and when we finished them. To distribute the tasks we used Kanban board scheme.

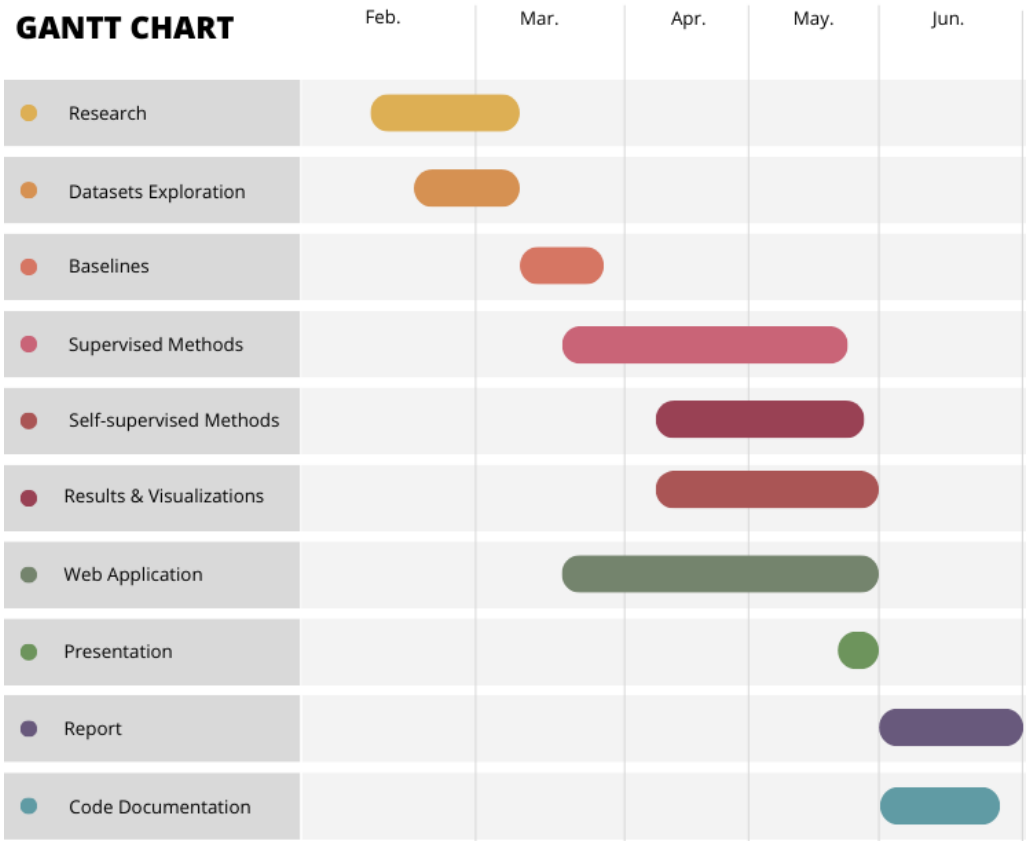


Figure 3: Simplified Gantt Chart of the Project

2.4 Ethical Aspects

Since all the data we use is already preprocessed and anonymized, we do not have access to explainability about the users or any sensitive information, which minimizes privacy concerns in our project. However, companies or third parties utilizing our approach with their own data must address several ethical considerations to ensure the responsible use of the technology. These ethical issues include:

Bias and Fairness: A key ethical concern is the potential for bias in the dataset and the model. If the training data contains biases, such as over-representation or under-representation of certain groups or behaviors, the model may repeat these biases, leading to unfair treatment for the users. For example, certain user demographics might be unfairly targeted as anomalous more frequently. Ensuring the data used is as unbiased and representative as possible and implementing fairness-aware algorithms are crucial steps to mitigate bias.

Privacy: Although our data is preprocessed and anonymized, reducing privacy concerns, companies using our approach with their own data must ensure data protection and privacy. This includes complying with data protection regulations such as GDPR, implementing strong data encryption, anonymization techniques, and informing users about how their data is being used.

Transparency and Accountability: AI models, particularly complex ones like GNNs, can function as "black boxes," making their decision processes opaque. Ensuring transparency in the model's decision-making process is vital for accountability. Techniques like explainable AI (XAI) can make AI decisions more understandable to humans, helping to build user trust and allowing users to contest decisions they believe to be incorrect.

False Positives and Negatives: In fraud detection, false positives (incorrectly identifying normal behavior as anomalous) can lead to unnecessary investigation and negative consequences for users, while false negatives (failing to identify actual fraud) can allow malicious activities to go undetected. Balancing these errors is crucial to maintain security and user trust. Regularly updating and validating the model with new data helps in minimizing these errors.

Ethical Use and Misuse: While the primary goal of developing such a system is to detect and mitigate fraudulent behavior, there is a potential risk of misuse. Organizations could use the technology for invasive surveillance or to unfairly target competitors or critics. Establishing clear ethical guidelines and usage policies, and ensuring that all stakeholders are aware of them, can help prevent misuse.

By proactively addressing these ethical aspects, companies can ensure that our graph anomaly detection system is used responsibly and benefits all stakeholders while minimizing risks.

3 Data

We utilize two primary datasets to evaluate the performance of our proposed architecture: Amazon Reviews and Yelp. Each dataset presents unique characteristics and challenges that are essential for comprehensive anomaly detection analysis.

3.1 Amazon Reviews Graph

The Amazon Reviews dataset[1] consists of user reviews on various products available on the Amazon platform. In this dataset, nodes represent users, and edges are formed based on specific interactions and similarities:

- Users who reviewed the same product.
- Users who gave the same rating to a product.
- High textual similarity between reviews.

Key characteristics of the Amazon dataset:

[1] https://paperswithcode.com/dataset/amazon-fraud

- Number of nodes: 11,944
- Number of features per node: 25
- Anomalous nodes: Users with less than 20% helpful votes
- Class distribution: Class 0 (Normal) - 90.5%, Class 1 (Anomalous) - 9.5
- Train: 60%, Validation: 20% and Test: 20%

3.2 Yelp Graph

The Yelp dataset[2] comprises user reviews on businesses listed on Yelp. Each review is represented as a node, and edges exist between nodes based on several criteria:

- Reviews by the same user.
- Reviews for the same product with the same rating.
- Reviews for the same product within the same month.

Key characteristics of the **Yelp** Reviews dataset:

- Number of nodes: 45,954
- Number of features per node: 32
- Anomalous nodes: Reviews considered spam
- Class distribution: Class 0 (Normal) - 85.5%, Class 1 (Anomalous) - 14.5%
- Train: 70%, Validation: 15% and Test: 15%

Both datasets are useful for training a graph anomaly detection model. However, the Amazon dataset is relatively easier as the node attributes allow the detection of anomalies directly, allowing almost any model to perform well and achieve good results, without the need to use the graph structure provided.

We primarily used the more challenging Yelp dataset to thoroughly evaluate our models' performance on the different experiments. This dataset presents a greater difficulty in distinguishing anomalies, providing a better evaluation of our model's capabilities and our training approach performance. Nevertheless, we conducted experiments using both datasets to ensure a more robust assessment of our proposed approach.

4 Methodology: Experiments, Learning and Training

In this section, we describe the various methods and techniques we employed in our study. Our approach encompasses a range of graph neural network backbones, different types of classification heads, and diverse training and learning strategies. Each subsection describes the specifics of these components, providing a thorough understanding of our methodology.

[2]https://paperswithcode.com/dataset/yelpchi

4.1 Graph Neural Network Backbones

Graph Neural Networks (GNNs) have emerged as a powerful tool for processing and learning from graph-structured data. These models leverage the inherent structure of graphs to perform tasks such as node classification, link prediction, and graph classification. In this subsection, we explore various GNN architectures, each with unique mechanisms for aggregating and learning from the graph data. The following sections provide a detailed overview of the four prominent GNN backbones we utilized in our experiments. You can check their visual intuition in figure 4.

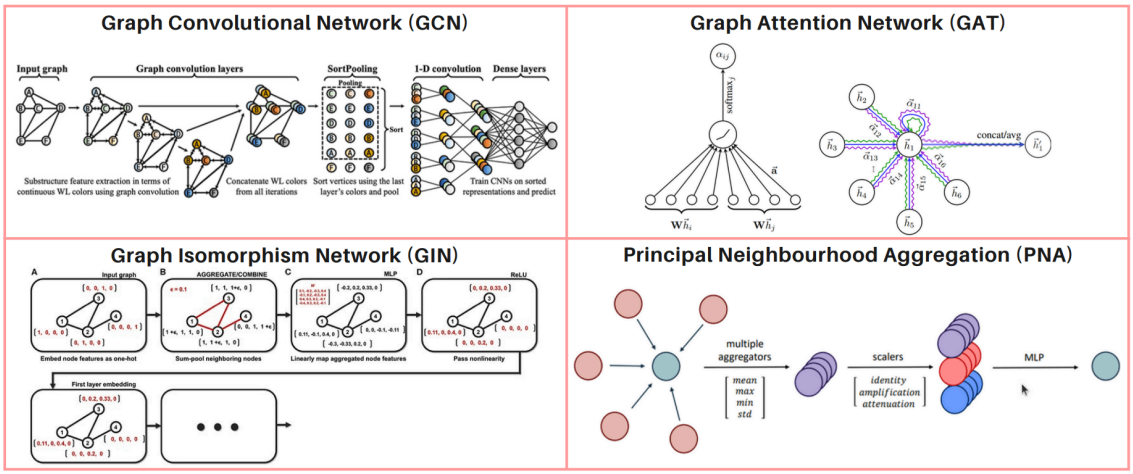


Figure 4: GNN Backbones Overview

4.1.1 Graph Convolutional Network (GCN)

Graph Convolutional Networks (GCNs) apply convolution operations to graph data, allowing for the aggregation of information from a node's neighbors. The key idea is to perform a weighted average of the feature vectors of neighboring nodes to capture local structural and feature information. This makes GCNs particularly effective for tasks like node classification and link prediction.

In our experiments with the Yelp and Amazon datasets, GCNs were used to learn node embeddings that capture the intrinsic properties and relationships between users and reviews. For example, in the Amazon dataset, nodes represent users and edges represent interactions based on shared reviews or similar ratings. By applying GCNs, we were able to aggregate features from neighboring nodes, helping the model to identify patterns indicative of anomalous behavior, such as fraudulent reviews or users.

4.1.2 Graph Attention Network (GAT)

Graph Attention Networks (GATs) introduce an attention mechanism to graph data, which assigns different weights to nodes, on the message passing, based on their importance. This attention mechanism allows the model to focus more on the most relevant nodes during the learning process, thereby enhancing its ability to detect subtle anomalies in the graph structure.

GATs were particularly useful for identifying influential nodes that have a significant impact on their neighbors. For instance, in the Yelp dataset, a review node connected to several influential users might receive higher attention weights, helping the model to better detect anomalies such as fake reviews that are connected to many legitimate users.

4.1.3 Graph Isomorphism Network (GIN)

Graph Isomorphism Networks (GINs) are designed to capture the isomorphism properties of graphs, meaning they can distinguish between different graph structures effectively. GINs employ a sum aggregation function, which has been proven to be more expressive than other aggregation methods. This makes them particularly powerful for tasks requiring a high level of structural awareness, such as detecting anomalies that deviate from normal patterns.

GINs were used to ensure that the learned embeddings accurately reflect the underlying graph structure. This was crucial for detecting anomalies that exhibit unusual structural patterns. For example, on Yelp graph, GINs helped to identify users or reviews that had unusual connections compared to the typical graph structure, indicating potential fraudulent activity.

4.1.4 Principal Neighbourhood Aggregation (PNA)

Principal Neighbourhood Aggregation (PNA) enhances GNNs by aggregating information from a node's neighbors using a combination of different aggregation functions and then combining them with MLP, performing a kind of ensemble network. This multi-aggregation approach allows PNA to capture a wide range of structural information, improving its robustness and performance in anomaly detection tasks.

PNA's use of multiple aggregation functions was particularly effective in capturing diverse structural features from the graphs, leading to more robust and expressive node embeddings. This helped the model to better distinguish between normal and anomalous reviews, thereby improving the overall detection performance.

4.2 Classification Heads

In our approach to anomaly detection on graphs, various classification heads were employed to map the learned node embeddings to anomaly scores. Each classification head has unique characteristics and advantages, which contribute to the overall performance of our model. In this subsection, we describe four different types of classification heads used in our experiments: Multi-Layer Perceptron (MLP), Gaussian Mixture Model (GMM) classifier, Threshold classifier, and other traditional Machine Learning classifiers.

4.2.1 Multi-Layer Perceptron (MLP)

The Multi-Layer Perceptron (MLP) is a type of feedforward artificial neural network that consists of multiple layers of nodes. Each node, or neuron, in one layer, connects with a certain weight to every node in the following layer. MLPs are capable of learning complex patterns in data due to their deep architecture.

In our experiments, the MLP was used as a classification head to map the node embeddings to anomaly scores. The architecture of the MLP included several hidden layers with activation functions such as ReLU (Rectified Linear Unit) and Batch normalization layers. The final layer used a softmax to produce a probability score indicating the likelihood of a node being an anomaly.

MLPs are particularly effective in capturing non-linear relationships in the node embeddings, thereby enhancing the model's ability to distinguish between normal and anomalous nodes.

4.2.2 Gaussian Mixture Model classifier (GMM)

The Gaussian Mixture Model (GMM) classifier is a probabilistic model that assumes all the data points are generated from a mixture of several Gaussian distributions with unknown parameters. GMM is particularly useful for modeling the distribution of node embeddings and identifying clusters that correspond to normal and anomalous nodes.

In our implementation, we decided to use a GMM to fit the distribution of the normal node embeddings, with a reduced dimensionality. Each node was assigned an anomaly score based on the likelihood of it being generated by the Gaussian components representing normal behavior. Once we had a GMM describing the normal data, we could find the threshold on the likelihood of deciding which nodes are flagged as anomalies.

We explored different options to reduce the node embedding dimensionality, but in the end, we decided to use the first layer of a simple two-layer MLP autoencoder, which provided the best results.

This GMM classifier is beneficial as it only models the normal data, so when detecting anomalies it theoretically should generalize better as the nodes flagged as anomalies are those that do not fit the usual distribution, not those within the distribution of anomalous nodes seen during training.

4.2.3 Threshold classifier

Based on the results seen on the GMM classifier approach, we thought that a single threshold on the node embeddings with a reduced dimensionality could perform as well, or better, removing steps that did not provide additional information in this case. An example of this can be seen in Figure 7.

So, this classification method was based on reducing the dimensionality of the embeddings to a single feature, with an MLP autoencoder as in the GMM classifier, and then searching for a threshold that could split correctly both types of data, normal and anomalous.

4.2.4 Machine Learning Classifiers

In addition to the MLP, GMM, and Threshold classifiers, we also experimented with various traditional Machine Learning classifiers to evaluate their effectiveness in our anomaly detection framework. These classifiers included:

- **Support Vector Machines (SVM)**: SVMs are powerful classifiers that work by finding the hyperplane that best separates the data into different classes. In our experiments, SVMs were used to classify node embeddings and were found to be effective in certain scenarios, particularly when the data was linearly separable at some dimension.

- **Random Forests**: Random Forests are ensemble learning methods that operate by constructing multiple decision trees during training and outputting the mode of the classes for classification tasks. They were used to leverage their robustness and ability to handle high-dimensional data in our anomaly detection framework.

- **k-Nearest Neighbors (k-NN)**: k-NN is a non-parametric method used for classification by comparing the distance between different data points. In our implementation, k-NN was used to classify nodes based on the similarity of their embeddings to those of labeled nodes.

Each of these classifiers contributed unique strengths to our anomaly detection system. For example, the SVM was particularly effective in the Yelp dataset for separating normal reviews from anomalous ones based on their node embeddings. Random Forests provided robustness in handling noisy data, while k-NN offered a simple yet effective approach for identifying anomalies based on nearest neighbors in the embedding space.

These diverse classification heads allowed us to thoroughly evaluate the performance of our anomaly detection framework and select the most effective methods for different types of graph-structured data, more discussion about specific results for each of the classification heads is done in section 5.

4.3 Training and Learning Approaches

In our experiments, we have compared different approaches to train our models, ranging from classical supervised learning to newer methods such as Self-Supervised Contrastive Learning. In this section, we discuss and explain each of the approaches in depth.

4.3.1 Supervised Learning

We began our experimentation with a supervised learning approach. Our goal was to train the entire model simultaneously, allowing it to encode the nodes and their positions in the graph into a single feature vector, which would then be used by the classification head. To achieve this, as we only had two classes, anomaly and normal, we used binary cross-entropy loss, which allowed us to train both the classification head and the preceding layers of the model together.

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p_i) + (1-y_i)\log(1-p_i)\right] \tag{1}$$

where N is the number of samples, y_i is the ground truth label for sample i (0 or 1), and p_i is the predicted probability of the node i being an anomaly.

One major challenge we faced with this approach was that our data was highly unbalanced. In both datasets, less than 15% of the nodes were labeled as anomalies. This imbalance could lead the model to perform poorly on the minority class (anomalies), as the model might become biased towards the majority class (normal nodes).

To address this issue, we employed a strategy to balance the loss function. Specifically, we adjusted the loss so that errors on the minority class (anomalies) were penalized more heavily than errors on the majority class (normal nodes). This adjustment makes the loss function more reflective of a balanced dataset, ensuring that the model pays appropriate attention to the minority class.

Balancing the loss in this way is more effective than simply undersampling the majority class because it allows the model to learn from all available data while still addressing the class imbalance. Undersampling could result in the loss of valuable information from the majority class, whereas our approach retains the full dataset.

The weight of each class in the loss function was calculated using the following formula:

$$W_c = \frac{N}{N_c^2} \tag{2}$$

where W_c is the weight for class c, N is the total number of nodes, and N_c is the number of nodes in class c. This formula ensures that classes with fewer examples (such as anomalies) are given more weight, thereby compensating for their scarcity in the dataset.

Building upon the supervised learning approach, we sought to improve the model's ability to learn effective embeddings by integrating supervised contrastive learning. By using both binary cross entropy loss and a contrastive loss simultaneously, we aimed to force the embeddings of nodes from the same class to be close together, while ensuring that nodes from different classes are far apart. This dual loss approach should result in better embeddings, which in turn would enhance the model's classification performance.

4.3.2 Supervised Contrastive Learning

To further improve our model, we explored using only supervised contrastive learning and afterwards train the classification head alone. This method aims to obtain more discriminative embeddings for classification. For this approach, we used triplet loss, a well-known loss function that ensures embeddings of nodes from the same class are closer together than those from different classes.

$$\mathcal{L}_{\text{triplet}} = \frac{1}{N} \sum_{i=1}^{N} \max\left(d(A_i, P_i)^2 - d(A_i, N_i)^2 + M, 0\right) \tag{3}$$

where A_i is the anchor point, P_i is the positive point, N_i is the negative point, d is a distance function (in our case cosine distance), M is the margin value, and N is the number of triplets in the batch.

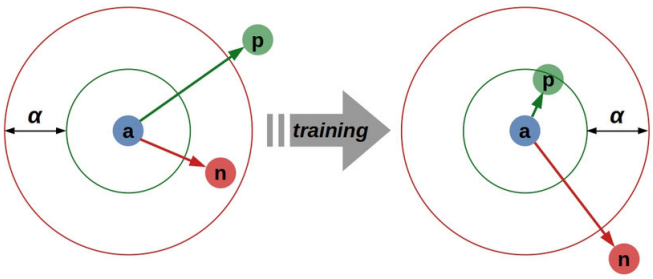


Figure 5: Triplet loss objective visualized.

4.3.3 Self-Supervised Contrastive Learning

In scenarios where we have a large amount of data but lack labeled instances, self-supervised contrastive learning can be particularly advantageous. This is often the case in anomaly detection tasks, where we may have numerous data points (e.g., reviews, user activities) without clear labels indicating anomalies, such as spam or unhelpful users.

To address this challenge, we investigated various self-supervised learning techniques suitable for our problem domain. Among these, we identified Graph Contrastive Learning (GraphCL) as a promising approach. GraphCL leverages graph augmentations and operates without the need for labels, training a model to produce similar representations for similar nodes through a contrastive learning framework.

The core idea behind GraphCL is to create two different augmented versions of the same graph. These augmentations involve randomly masking some node attributes and removing random edges based on a Bernoulli distribution. The model is then trained to maximize the

similarity between the representations of the same node in these two augmented versions while minimizing the similarity between the representations of different nodes. This process is illustrated in Figure 6.

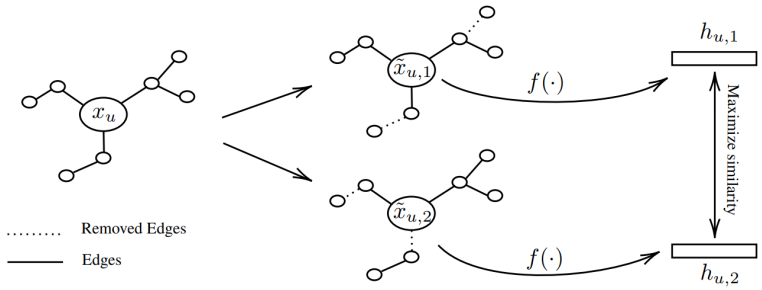


Figure 6: Example of the GraphCL pipeline.

To implement this approach, we utilized the InfoNCE (Information Noise Contrastive Estimation) loss function, which is defined as:

$$\mathcal{L}_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(z_i, z_k)/\tau)} \tag{4}$$

where z_i and z_j represent the embeddings of the positive pair (i.e., the same node in different augmentations), $\text{sim}(z_i, z_j)$ denotes the similarity function (such as the dot product or cosine similarity), τ is a temperature parameter that scales the similarity scores, N is the total number of nodes, and $\mathbb{1}_{[k \neq i]}$ is an indicator function that equals 1 if $k \neq i$ and 0 otherwise.

Through this method, the model learns to produce embeddings where nodes with similar attributes and connections are close to each other in the embedding space, while nodes that are dissimilar are far apart. This capability is particularly useful for detecting anomalies, as anomalous nodes are expected to have representations that are distinct from those of normal nodes.

We also conducted experiments with variations of the augmentation strategies described in the GraphCL paper. One notable observation was that completely masking some node attributes was not realistic, as it would be unusual to encounter a value of 0.0 for an attribute in our processed graphs. Instead, we experimented with adding random noise to these attributes, which we found to be a more realistic and effective augmentation strategy. This approach preserves the natural variability in the data while still enabling the model to learn robust representations that are effective for anomaly detection.

4.3.4 Graph Autoencoders

Lastly, we explored Graph Autoencoders as a self-supervised method for learning node representations. Autoencoders are a type of neural network used to learn efficient codings of unlabeled data, making them well-suited for tasks where labels are scarce or unavailable. In our experiments, we focused on two variants of graph autoencoders: reconstruction autoencoders and edge autoencoders.

Reconstruction Autoencoders

The first variant we experimented with was reconstruction autoencoders. In this approach, the goal is to train a model to accurately reconstruct the original attributes of the nodes

from a compressed representation. This involves an encoder-decoder architecture, where the encoder compresses the node attributes into a lower-dimensional "bottleneck" representation, and the decoder reconstructs the original attributes from this bottleneck.

The key idea is that the bottleneck forces the model to learn meaningful and compact representations of the nodes that capture the essential features needed for reconstruction. The node representations, therefore, are the embeddings obtained at the bottleneck layer of the model.

The loss function used in this approach is the mean squared error (MSE), which measures the difference between the original node attributes and their reconstructed counterparts:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{D}\sum_{i=1}^{D}(y_i - \hat{y}_i)^2 \tag{5}$$

where D is the dimensionality of the vectors, y_i is the ground truth vector, and $\hat{y}_i$ is the predicted vector.

We also experimented with different variations of this reconstruction approach:

- *Using only the normal class nodes:* This variation involved training the autoencoder exclusively on nodes from the normal class. The rationale was that the autoencoder would learn to reconstruct normal nodes accurately, leading to higher reconstruction errors for anomalous nodes.

- *Reducing the reconstruction error on the normal class, maximizing them on the anomaly class:* In this variation, we aimed to explicitly minimize reconstruction errors for normal nodes while maximizing them for anomalous nodes. This approach sought to make the autoencoder particularly sensitive to anomalies.

Both variations aimed to achieve higher reconstruction errors for anomalous nodes compared to normal nodes. After training, we could classify a node as an anomaly based on its reconstruction error: higher errors would indicate anomalies. The results shown on the section 5 correspond to the second variation explained.

Edge Autoencoders

The second variant we explored was edge autoencoders. Instead of focusing on node attributes, edge autoencoders aim to learn node representations by predicting the existence of edges between nodes. This approach leverages the structural information in the graph to inform the embeddings.

In edge autoencoders, the model is trained to predict whether a pair of nodes is connected by an edge. This involves learning embeddings that capture both the local and global structure of the graph. For each node pair, the model predicts the likelihood of an edge existing between them.

To implement it, we made a modification to our base model by incorporating an attention module in the classification head. The attention module helps to focus on the most relevant parts of the graph structure when making predictions. During training, we trained the node embeddings using three separate backbones, each trying to predict their corresponding edges.

Later, when training the classification head, we also trained the attention module to combine the information extracted by each of the backbones. This multi-faceted approach allows the model to learn comprehensive and robust node representations by integrating information from various structural perspectives within the graph.

Both the reconstruction and edge autoencoders provide valuable insights into the node representations, enhancing our ability to detect anomalies effectively.

5 Results

5.1 Metrics

To report the results, we have chosen four metrics. Two of them represent the general classification capabilities of the model (AUC and Macro F1), and the other two are specific for the performance on anomaly detection (Precision and Recall of anomalies).

The metrics used are the following:
AUC The Area Under the Receiver Operating Characteristic Curve (AUC) helps evaluate a model's ability to distinguish between positive and negative classes. It summarizes the ROC curve, which plots the true positive rate (correctly identified positives) against the false positive rate (incorrectly identified positives) across all classification thresholds. So, this metric allows us to have a general overview of the classification performance of the model.

Macro F1 Macro F1 is a metric used in multi-class classification problems. It tells you how well a model performs on average across all the different classes in your data. Unlike the overall F1 score, which can be skewed by imbalanced datasets, Macro F1 treats each class equally, important in this case as the anomalies represent less than 15% of the instances. It computes the F1 score for each class individually, and then simply takes the average of those scores. F1 is a good metric as it takes into account both precision and recall of the model.

Precision Precision measures the accuracy of the model's positive predictions. It tells you what proportion of the things your model identified as positive were actually correct. It's calculated by dividing the number of true positives (correctly identified positive cases) by the total number of positive predictions (both true positives and false positives).

In this section when we say precision it corresponds to the model precision on anomalies.

Recall Recall measures the completeness of the model's positive predictions. It tells you what proportion of the actual positive cases did your model identify? It's calculated by dividing the number of true positives by the total number of actual positive cases (including those the model missed, called false negatives).

In this section when we say recall it corresponds to the model recall on anomalies.

5.2 Classification Heads

In this comparison, we wanted to know which classification head worked better for this task considering our proposed architecture. In table 1 can be seen the performance of the model trained on our supervised approach for the Yelp graph with different classification heads. We decided to show the comparison on the Yelp graph as detecting anomalies in this case is a more difficult task which provides more differences in performance on the different methods.

We can see that each model performs better in one of the metrics, but the MLP head, which was trained at the same time as the model generating the embeddings, has the most

stable performance across the different metrics. For this reason, it is the performance that we will use during the following comparisons. It is worth noticing that those classifiers that were thought to separate anomalous and normal nodes by modeling the normal distribution have similar performances to the MLP, achieving the highest recall of anomalies using the Threshold classifier.

Classification Head	AUC	Macro F1	Precision	Recall
MLP	**94.49**	84.74	68.88	80.64
Threshold	94.48	84.59	67.91	**81.54**
GMM	91.89	84.52	**81.53**	74.00
SVM	94.45	**85.05**	77.32	71.46

Table 1: Comparison of classification heads using Yelp graph, PNA model as the backbone, and the model producing the embeddings trained on the supervised learning approach.

We think that GMM classifier does not achieve higher performance because it is an additional step in the classification that does not add many insights, as we can see looking at the performance of directly applying a Threshold to the embeddings with a reduced dimensionality. This can be seen in Figure 7 in which a single threshold (Threshold classifier) allows a good classification between normal distributed nodes and anomalous ones. We think that this classifier, GMM, could perform better in more complex cases, in which we could not model the distribution of normal class with a single feature.

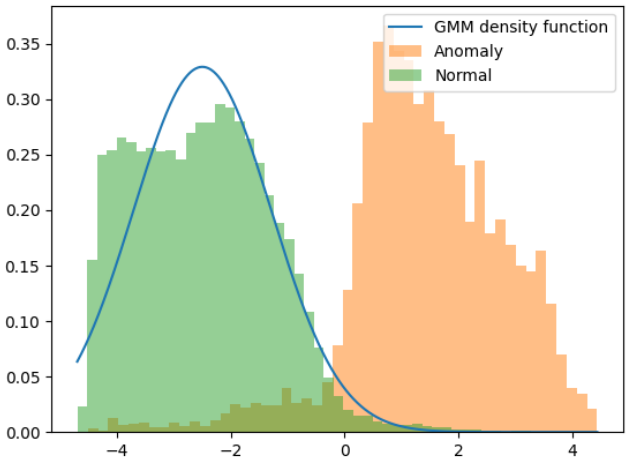


Figure 7: Normalized distributions of anomalous and normal node embeddings of the compared model, after reducing the Dimensionality with an MLP autoencoder. It also shows the GMM density function of the normal data.

5.3 Backbones

We wanted to test our proposed architecture using different graph neural networks as the backbone, in order to make this comparison we decided to use our Supervised Approach, as in the previous comparison and for the same reason we used the Yelp graph. The classification head used to report the metrics is the MLP trained at the same time as the backbones.

In table 2, we can see that one of the backbones with the simplest message passing, PNA, is the one that performs better for our pipeline. We think that this is caused as it produces a kind of ensemble network, using multiple message passing functions, which allows the model to represent better the structure of the graph. Also, the better performance of GIN with respect to GAT could mean that the structure and situation of a node inside the graph might be relevant in order to detect anomalies in Yelp graph, as GIN performs better in all the metrics.

Backbone	**AUC**	**Macro F1**	**Precision**	**Recall**
GAT	91.04	80.69	63.88	71.15
GIN	92.49	82.46	67.46	73.25
PNA	**94.49**	**84.74**	**68.88**	**80.64**

Table 2: Comparison of backbones using Yelp graph and our supervised learning approach.

5.4 Training Approaches

Once we established the best parameters for the model, which was the best backbone and classification head, we wanted to compare how performance was affected by the use of different training approaches. In this section, we will show the ablation study done for both Graphs, regarding the training approaches.

In order to make this comparison we use PNA as the backbone, as it was the best performing one, and MLP as the classification head, which was the one that provided more stable results across different metrics. We show the best results accomplished for each of the training approaches.

5.4.1 Yelp Graph

In table 3 it can be seen the results of the ablation study. If we look at AUC and Macro F1 we can see that the best-performing approach has been our supervised learning approach, which combines cross entropy loss and triplet loss, this means that it was the model that performed better classification without taking into account the classification threshold and using the standard 0.5.

It is important to notice that within the self-supervised methods to train the embeddings, we only achieved embeddings which allowed to detect anomalies with the edge autoencoder. We think that the other three methods failed on this dataset for the same reasons. We consider that the contrastive self-supervised learning failed as we were training a model that theoretically was invariant to slight changes in the node connections and node attributes, but it could be the case that in order to differentiate those nodes in this graph these slight changes are important.

Approach	AUC	Macro F1	Precision	Recall
ML Baseline	93.08	81.07	**86.11**	54.81
Contrastive Supervised Learning	92.26	80.09	57.12	81.63
Supervised Learning	**94.49**	**84.74**	68.88	80.64
Contrastive Self-Supervised Learning (Drop.)	54.82	53.38	24.07	13.57
Contrastive Self-Supervised Learning (Samp.)	59.57	49.56	26.88	50.89
Reconstruction Autoencoder	52.57	43.22	15.19	53.29
Edge Autoencoder	87.07	68.92	38.14	**82.14**

Table 3: Comparison of approaches using Yelp graph, PNA model as backbone and an MLP as classification head. ML Baseline uses of a Light Gradient Boosting Machine classifier, which achieved the best performance out of several ML algorithms.

Taking a look into the performance in anomaly detection, we can see the Recall and Precision. The best precision was obtained on the ML baseline, this could be caused by the low recall of anomalies obtained on this approach, so the model only was capable of detecting those more clear cases. But, the second best performing model on precision was also the supervised learning approach. In terms of recall, the best performance was achieved on the edge autoencoder, followed nearby by the supervised approach.

This good recall and worse precision of the node autoencoder approach can be seen if we look at the embeddings generated by the model, Figure 8, as the anomalous reviews nodes (Spam) are within a clear cluster, but this cluster is superposed to the normal nodes making it impossible to have great precisions.

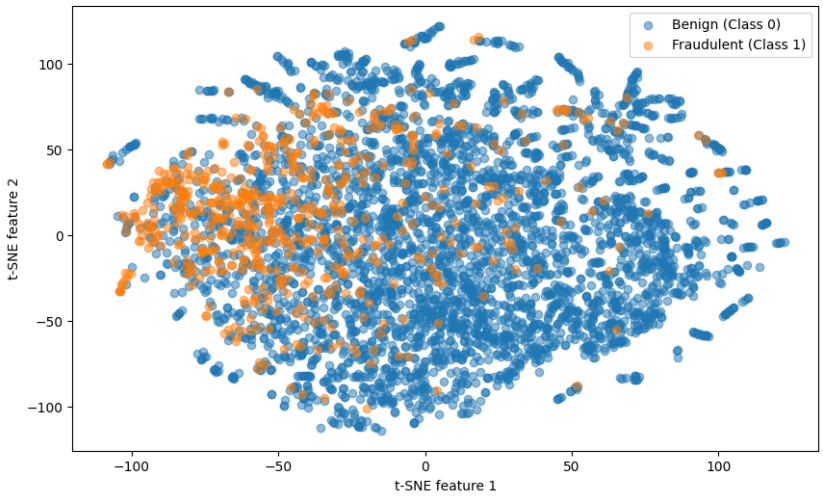


Figure 8: Node representations for the normal ones, blue, and anomalous ones, orange, given an edge autoencoder model.

Finally, in Figure 9 it can be seen the best approaches comparison, with the addition of the state of the art, in a more visual way. It is important to notice that with our approach we achieved a performance close to the state of the art in the AUC score, ours being 94.49 and the SoTA 94.98.

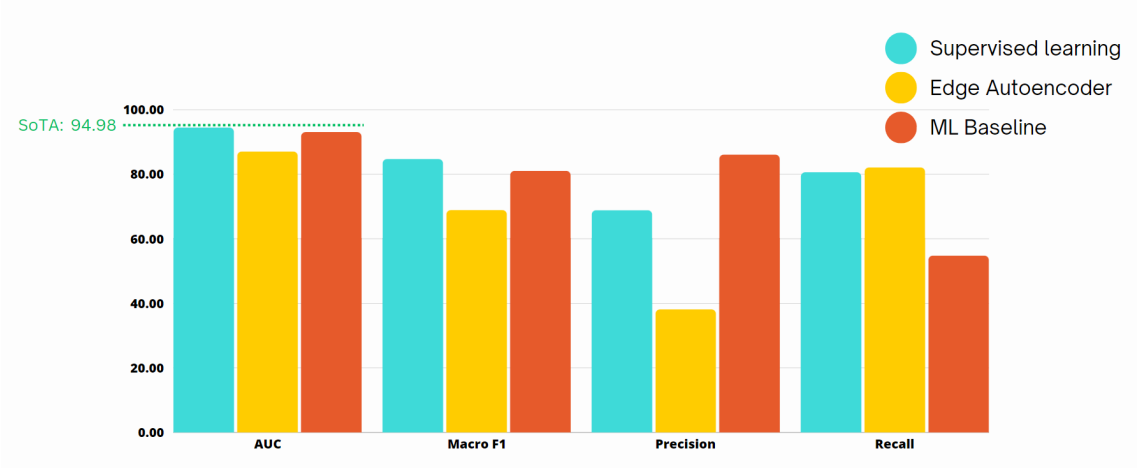


Figure 9: Visual comparison of the best-performing approaches and the State of the Art.

5.4.2 Amazon Graph

We replicated the same study on the Amazon Graph, but in this case, the results were not as good. In Table 4 we can see that the ML baseline outperformed each one of our methods. This is caused due to the highly processed attributes of the nodes on this dataset. As it can be seen in Figure 20 the original features of the anomalous nodes already allowed an easy classification from the rest of the nodes. In consequence, the addition of the graph structure of the data did not add useful information in our case. For these reasons, in order to decide the best training method we used the information obtained on the Yelp graph.

In the results table of this experiment, we can see similar tendencies to those in the Yelp graph. Supervised methods are the best-performing ones in a general classification scope (Macro F1) and detecting anomalies (Precision and Recall of anomalies). As before, the edge autoencoder accomplished good recall of the anomalies, but a bad precision for the same reasons.

As an exception, in comparison with the Yelp graph, we accomplished a closer performance to the rest of the methods with the self-supervised contrastive method, this happens due to the better starting features, already allow detecting anomalies.

Approach	AUC	Macro F1	Precision	Recall
ML Baseline	**97.57**	**92.80**	**96.05**	79.26
Contrastive Supervised Learning	93.80	89.37	78.28	83.53
Supervised Learning	93.21	88.27	78.31	79.26
Contrastive Self-Supervised Learning (Drop.)	91.93	71.38	37.16	**84.75**
Contrastive Self-Supervised Learning (Samp.)	89.93	76.53	53.33	63.14
Reconstruction Autoencoder	83.87	86.95	90.00	65.85
Edge Autoencoder	95.12	84.60	65.51	81.10

Table 4: Comparison of approaches using Amazon graph, PNA model as backbone and an MLP as classification head. ML Baseline uses of an Ada Boost classifier, which achieved the best performance out of several ML algorithms.

5.4.3 Data Efficiency

After seeing that our Supervised Learning approach was the best performing one for this task in our ablation study, we wanted to experiment with how the performance varies depending on the amount of data available to train. This experiment was important as it is a domain with a restricted amount of labeled data, which our approach requires, knowing the amount of data needed is important.

In this experiment we used the same test and validation set as before, but we made a subsample of the training data used in the experiments which used less than 70% out of the total amount of data. For this experiment, we used PNA as the backbone with an MLP classification head and the different models were trained with our supervised learning approach. To evaluate this experiment we used two metrics, Average precision and Area under the curve, that evaluate the model performance classifying for all the different classification thresholds not only for a single one.

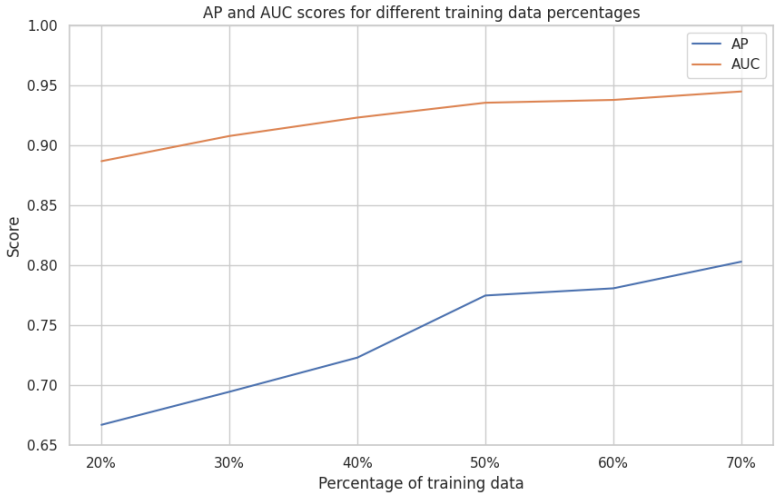


Figure 10: Average precision (AP) and area under the roc curve (AUC) evolution as we use more data to train the model. This figure was made using PNA as backbone and our supervised learning approach.

From Figure 10 we can see that the different models perform well even with a very low percentage of data used to train the model. We can also see a peak of performance from 50% onwards, achieving results near the performance with the entire train set, 70%. So, we can assume that our model architecture and model architecture could be trained on other graphs with less amount of labeled data available and continue achieving great results.

5.5 Proposed Architecture

An important experiment was to check if our proposed architecture actually improved results compared to using a more usual architecture, in which we used a single graph neural network backbone that knew from which class was each edge.

This comparison was made using the best-performing setup given our previous ablation study, our Supervised Approach with PNA as backbone and MLP as classification head, compared to using a single PNA which used all different edges types without any attention mechanism of Self Attending edges, but we provided the labels of each edge using label encoding.

In Table 5 we can see the performance comparison of both architectures. In this comparison, our architecture outperforms on each metric the single backbone one. We can see that especially on those metrics specific to the anomaly class the difference in performance is bigger, with almost a 10% difference in precision and recall of anomalies.

This performance difference makes us think that this architecture allows a better understanding of the node situation on the graph. We think that in this case is quite relevant as the different types of connections are very different between each other and the separate processing allows for gaining more insightful features for each case.

Architecture	AUC	Macro F1	Precision	Recall
Single PNA	90.42	79.39	59.59	72.55
Self Attending Edges	**94.49**	**84.74**	**68.88**	**80.64**

Table 5: Comparison of the proposed architecture vs using a single PNA that treats the different types of edges at the same time, both models were trained using our supervised learning approach.

5.6 Analysis Best Models

5.6.1 Yelp Graph

As we have seen in the previous ablation study, the best-performing model on the Yelp graph was trained using our supervised learning approach, PNA as graph neural network backbone and MLP as classification head. In this section, we look into more detail of this model's performance.

Quantitative results

In Figure 11 it can be seen the confusion matrix of this model, in which we can visually see the good recall of the anomalies as well as the good performance in terms of the reviews predicted as an anomaly.

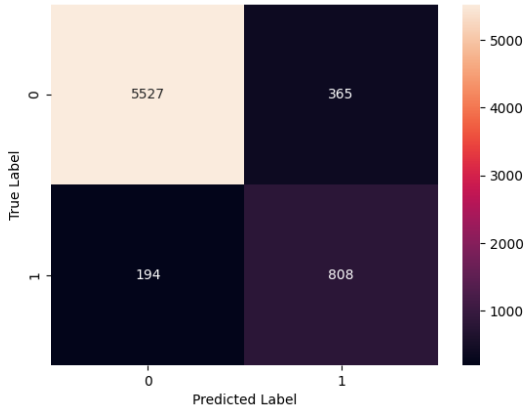


Figure 11: Best Yelp model confusion matrix. Class 0 corresponds to normal nodes and class 1 to anomalies.

We have also computed the ROC and Precision-Recall curves (Figures 12 and 13) to visualize the model performance across different thresholds. In both curves, we can see the

good performance of the model, especially in the ROC curve being close to the ideal 100% true positive rate and 0% of false positive rate point.

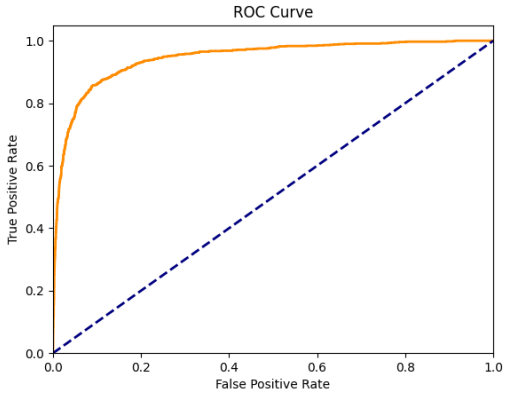


Figure 12: Best Yelp model ROC curve.

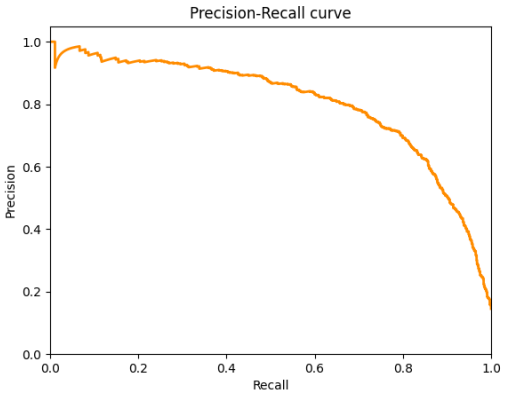


Figure 13: Best Yelp model Precision-Recall curve.

Qualitative results

In Figures 14 and 15 we can see a comparison between the original model attributes and the generated embeddings of our model, for the test set. In these figures it can be seen that the original features for the anomalous nodes were distributed around all the feature space, making it more difficult for the posterior classification step, instead on the learned model the representations of the anomalous ones produce a cluster speared from the rest of nodes.

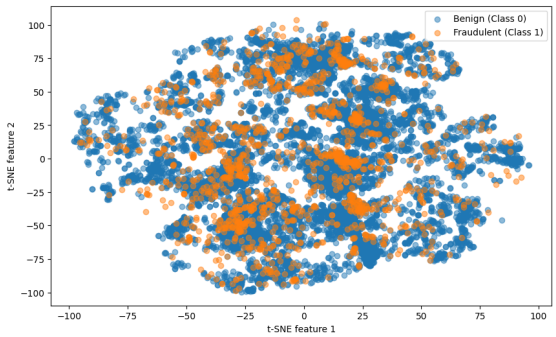


Figure 14: Yelp node attributes visualization using t-SNE.

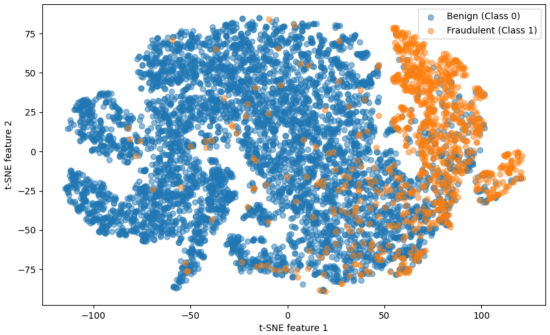


Figure 15: Yelp node model embeddings visualization using t-SNE.

The last analysis made was about the behavior of the attention module included in our model architecture. In Figure 16 it can be seen how the weights are distributed on the three types of edges and for each class of node.

The second and third types of edges (Reviews under the same product with the same star rating and reviews posted by the same user) are similarly distributed for both types of reviews, instead, in the first edge type (Two reviews from the same product) we can see a very clear distinction between the distributions. We can see that for this connection those anomalous reviews consistently have higher weights, which could mean that the other

reviews of the product might be important to detect anomalous reviews. We think that this could be caused as it might be that those products that contain anomalous reviews, spam, usually contain more than one of these kinds of reviews.

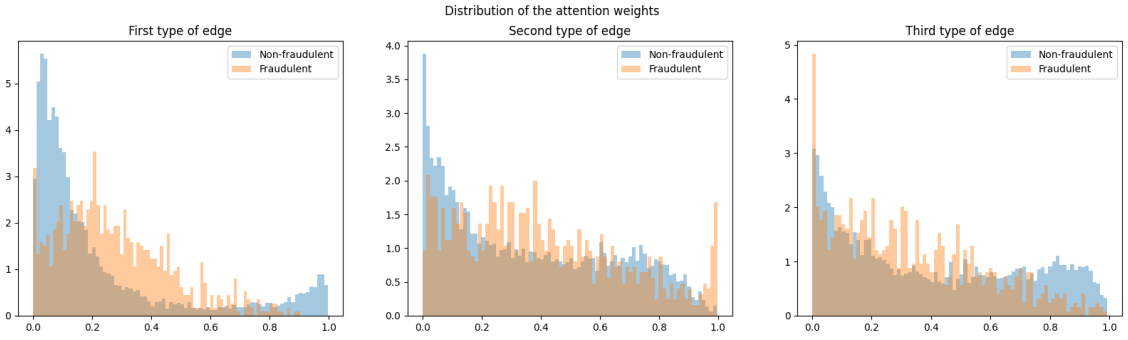


Figure 16: Attention weights distribution, of the attention module in our architecture, for the different test nodes.

5.6.2 Amazon Graph

As we have seen in the previous ablation study, the best performing model, which uses the graph structure, on Amazon graph was trained using the contrastive supervised learning approach, PNA as graph neural network backbone and MLP as classification head. In this section, we look into more detail of this model performance.

Quantitative results

In Figure 17 it can be seen the confusion matrix of this model, in which we can visually see the good recall and precision over the anomaly detection.

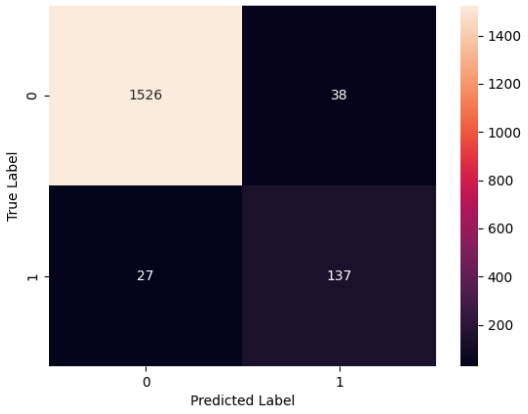


Figure 17: Best Yelp model confusion matrix. Class 0 corresponds to normal nodes and class 1 to anomalies.

We have also computed the ROC and Precision-Recall curves (Figures 18 and 19) to visualize the model performance across different thresholds. In the ROC curve, we can see the good performance of the model, reaching almost 80% of true positive rate with almost

0% of false positive rate. In the precision and recall curve, it can be seen the good precision that the model archives until the 80% recall.

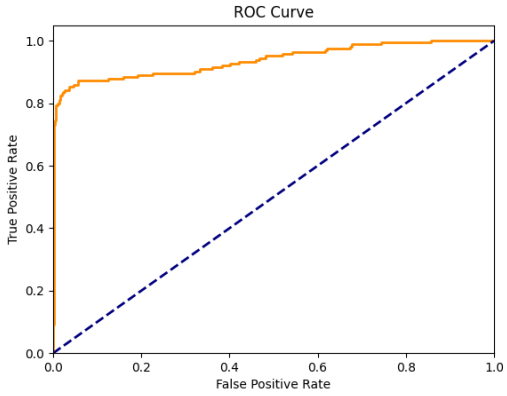


Figure 18: Best Amazon model ROC curve.

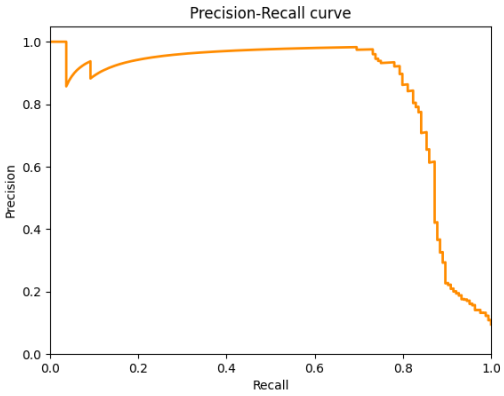


Figure 19: Best Amazon model Precision-Recall curve.

Qualitative results

Lastly, in Figure 20 and 21 we can compare the generated embeddings by our model and the original ones. It can be seen that the original attributes already separated those anomalous nodes from the rest, with the exception of a small amount of nodes distributed around. In the model-generated embeddings, it seems that the amount of anomalous nodes distributed within the normal nodes is reduced a bit, but it cannot be seen as a huge improvement. This is reflected in having no improvement in performance with respect to a model that classifies directly from the original attributes.

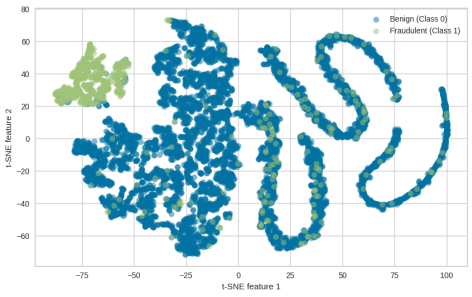


Figure 20: Amazon node attributes visualization using t-SNE.

Figure 21: Amazon node model embeddings visualization using t-SNE.

5.7 Conclusions of Our Contribution

Summing up the results of the experimentation, we can see that the proposed architecture improves the performance of using a single graph neural network backbone for this task. Also, in the Yelp dataset, we can see that the connection between different reviews is relevant in order to detect the anomalous ones, as we see that we achieve an improvement in recall without penalizing the rest of the metrics.

In terms of the best performance achieved, was obtained in our supervised learning approach, which combines the use of triplet loss and binary cross entropy leading to better

node representations and better performance detecting anomalies. Moreover, even though it is a supervised learning method we have shown that it is not a data-hungry model achieving similar results with much less training data.

5.8 Insights About the Model

In the evaluations done of the model, we can see that it would be reliable when used to detect spam or unuseful users, as we achieve high recall of these anomalies as well as good precision of the predicted anomalies.

Related to the scalability of our pipeline, it is scalable to the use of bigger graphs, and it could be used on other domains related to node classification. The scalability is restricted if the graph used has a lot of different types of connections that could lead to much higher complexity, in terms of the number of required parameters, this problem could be assessed in further work.

An important insight of our model is that after training, on GPU, it can make inference on CPU in real time, obtaining the results instantly. So, in order to achieve this important feature it is not necessary the use of GPU and in consequence a more complex infrastructure at inference time.

6 The Spotter

The Spotter is a web app that was born under the motto "See beyond the graph" with the intention to provide not only model predictions but also a user-friendly interface capable of displaying the results of the models in a visual way.
The Spotter welcomes the user with a page where 3 buttons can be seen on the top part of the screen and a GitHub logo on the top left. By clicking the GitHub logo, the user will be redirected directly to the project's GitHub repository. The other 3 buttons are, "Home", "About Us" and "Playground" as seen in the following figure.

Figure 22: Home page of The Spotter

The "About Us" button directs users to a brief description of the project, providing an overview of its purpose and features. Meanwhile, the "Playground" button takes users to an interactive space where they can engage with the app's core functionalities.

Upon entering the Playground, users will notice a sidebar that can be clicked to reveal an options panel. Within this panel, users have the choice to upload a graph by selecting either the "Display Amazon Graph" or "Display Yelp Graph" options. After uploading a graph, users can then run it through a model by clicking on the corresponding model button.

Once the model has processed the graph, the results are displayed for the user. To enhance visualization and ensure clarity, users can enable the physics of the graph, preventing the nodes from overlapping and providing a more coherent and comprehensible view of the data. This feature allows users to better understand the relationships and insights derived from the graph, fulfilling The Spotter's promise to help users see beyond the graph.

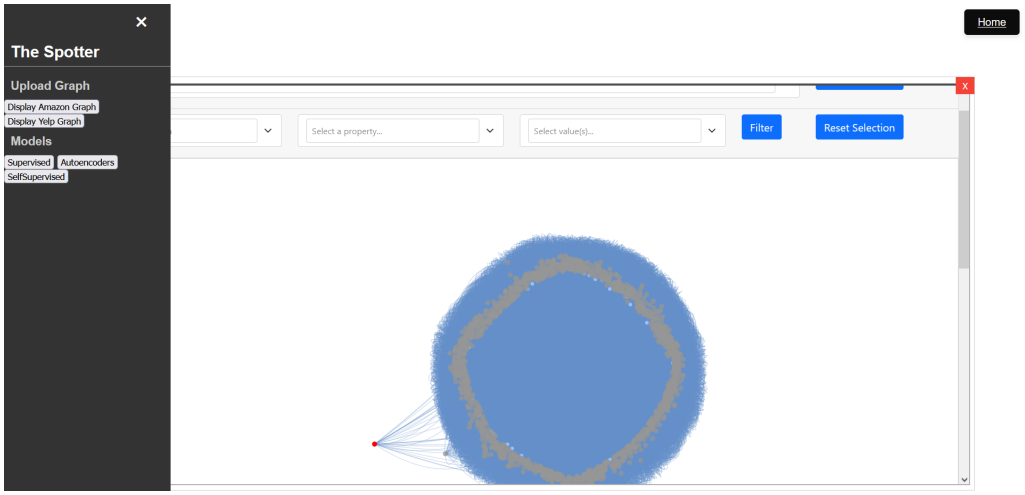


Figure 23: Playground section with loaded graph

When the model results are loaded onto the graph, hovering the mouse over a node reveals not only the node's ID but also its anomaly ratio. No additional node attributes are displayed, as they are not deemed meaningful.

Using the web app is straightforward. Navigate to The Spotter folder and execute the app.py file. This will generate an HTTP address; clicking on this address opens the app in a new browser tab.

The Spotter can be an invaluable tool for companies, streamlining the often complex and time-consuming process of detecting bots within their networks. By providing a more intuitive and user-friendly interface, The Spotter enables IT and security teams to quickly identify and analyze suspicious activity. This not only enhances the efficiency of bot detection but also allows organizations to proactively manage and mitigate potential threats. With its visual representation of data and straightforward interaction, The Spotter transforms the way companies handle network security, ultimately contributing to a safer and more secure digital environment. Additionally, by reducing the manual effort required for bot detection, companies can allocate their resources more effectively, focusing on other critical areas of their operations.

7 Discussion and Future Work

7.1 Implications of Our Work

The implications of our work extend across various domains, offering significant improvements in data integrity, trustworthiness, and consumer protection if specifically applied to

item review datasets such as Amazon or Yelp. Our anomaly detection framework, which leverages Graph Neural Networks (GNNs) and proposes a new edge self-attention strategy, has shown to be effective in several key areas that are of much interest to companies like Amazon, its users and the community as a whole:

Enhancing Dataset Quality: By improving the detection of anomalies, our approach enhances the overall quality of datasets. Higher quality datasets lead to better performance of machine learning models and more reliable research outcomes. In the context of online reviews, this means that data used for training and analysis is more accurate and trustworthy, reducing the noise introduced by fraudulent activities.

Improving Recommender Systems: Our model provides users with more accurate and reliable product or service recommendations. By identifying and filtering out fake reviews and anomalies, recommender systems can offer better suggestions, enhancing user satisfaction and trust in the platform.

Protecting Businesses: Businesses can protect their brand image and maintain customer trust by identifying and mitigating fraudulent activities. This protection extends beyond financial transactions to include the integrity of customer reviews and interactions, which are crucial for maintaining a positive brand reputation.

Market Insights and Consumer Behavior: Businesses can make more informed decisions based on accurate data insights. Our model enables a deeper understanding of consumer behavior by ensuring that the data used for analysis is free from anomalies. This leads to better products and services, ultimately benefiting both consumers and businesses.

7.2 Domain Adaptation and Transfer

Our approach allows for easy domain adaptation and transfer, given a low number of edge types. The flexibility of our model to adapt to various domains can be highly beneficial in several scenarios:

Financial Fraud Detection: Our model can identify unusual or suspicious patterns in financial transactions, which is crucial for preventing fraud in banking and financial services.

Transport Networks: In transport networks, our model can detect overloaded stations or other anomalies in the metro network, ensuring better service management and safety.

Autonomous Vehicles: Detecting anomalies in sensor networks used by autonomous vehicles can enhance safety and reliability, preventing accidents caused by sensor failures or malicious interference.

Social Networks: Our model can detect fake accounts and spam bots based on interaction patterns, which is essential for maintaining the integrity of social media platforms.

Recommender Systems: In recommender systems, detecting anomalous behaviors in user-item interaction graphs can improve recommendation accuracy and user satisfaction.

Biological Networks: Our approach can be adapted to detect malfunctioning neurons in biological networks, providing assistance in medical research and the development of treatments for neurological disorders.

7.3 Future Work

While our model has shown great promise, there are several areas for future work to further enhance its capabilities and address current limitations:

Scalability with Edge Types: One of the primary challenges in our model is managing the exponential increase in computational complexity when handling a large number of distinct edge types. Each edge type necessitates a unique self-attention mechanism, which significantly increases the computational cost. Specifically, for E different edge types, the complexity of our architecture can be expressed as:

$$\mathcal{O}(E \cdot N^2) \tag{6}$$

where N represents the number of nodes in the graph. This quadratic relationship shows the scalability issue, particularly as the number of edge types increases, as they directly affect the number of GNNs used as backbone.

To address this, we propose leveraging dimensionality reduction techniques to identify and retain the most informative edge types. One such technique is Principal Component Analysis (PCA), which can reduce the dimensionality of the edge-type feature space while preserving the maximum variance. This reduction is crucial for maintaining the model's performance while mitigating computational costs. The dimensionality reduction process can be formalized as:

$$\mathbf{X}' = \mathbf{X}\mathbf{W} \tag{7}$$

where $\mathbf{X}$ is the original matrix of edge type features, $\mathbf{W}$ is the matrix of principal components, and $\mathbf{X}'$ represents the transformed edge types with reduced dimensionality.

The selection of edge types can be further optimized by evaluating the contribution of each edge type to the overall variance. By selecting the top k principal components, we ensure that the reduced set of edge types captures the most significant information, which can be expressed as:

$$\mathbf{X}' = \mathbf{X}\mathbf{W}_k \tag{8}$$

where $\mathbf{W}_k$ consists of the top k principal components. This approach not only reduces the computational complexity but also helps in maintaining or even improving the model's performance by focusing on the most informative edge types, eliminating those that might be redundant or even confuse the model.

Additionally, other techniques such as feature selection algorithms can be explored to identify and retain the most significant edge types, further enhancing the scalability of the model.

Other options that could be interesting if those different types of connections do not have attributes, like on both graphs used in this project, could be to make an analysis of them and select those types of connections that mean similar things to process them together. So instead of training a GNN for each connection type use multiple connection types in each of the GNNs using adding an attribute, as a one-hot vector, to the connection which allows the model to identify that not all edges mean the same.

Optimizing Self-Attention Mechanisms: Optimizing the self-attention mechanisms within our model is essential to enhance efficiency and reduce computational overhead. The self-attention mechanism, which computes attention scores for each pair of nodes, can be formalized as:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ik})} \tag{9}$$

where e_{ij} is the attention coefficient between node i and node j, and $\mathcal{N}(i)$ represents the neighborhood of node i. This formulation allows the model to focus on the most relevant nodes, but the computational cost can be high, particularly for large graphs with many edge types.

To optimize this process, several strategies can be adopted in our future work:

- **Sparsity-Inducing Regularizations:** One approach to reduce computational overhead is to introduce sparsity-inducing regularizations. These regularizations encourage the model to assign higher attention scores to a smaller subset of edges, effectively reducing the number of non-zero α_{ij} values. This can be achieved by adding a sparsity constraint to the loss function:

$$\mathcal{L}_{sparse} = \mathcal{L}_{original} + \lambda \sum_{i,j} |\alpha_{ij}| \tag{10}$$

  where $\mathcal{L}_{original}$ is the original loss function, λ is a regularization parameter, and $\sum_{i,j} |\alpha_{ij}|$ encourages sparsity in the attention scores.

  If we had fewer non-zero α_{ij} values, the computations would be done between fewer elements, leading to a lower computational overload.

- **Hierarchical Attention Mechanisms:** Another approach is to implement hierarchical attention mechanisms, which aggregate information at different levels of granularity. This hierarchical approach can reduce the computational cost by first summarizing information at a local level and then aggregating these summaries at a higher level. The hierarchical attention can be expressed as:

$$\alpha_{ij}^{(l)} = \frac{\exp(e_{ij}^{(l)})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ik}^{(l)})} \tag{11}$$

  where $\alpha_{ij}^{(l)}$ and $e_{ij}^{(l)}$ represent the attention coefficients and scores at level l. This multi-level aggregation helps in reducing the overall number of computations while retaining essential information during the edge self-attending process.

- **Efficient Attention Score Computation:** Improving the efficiency of attention score computation can be another critical area. Techniques such as low-rank approximations can be used to approximate the attention scores more efficiently. For instance, the attention scores can be approximated using matrix factorization:

$$\mathbf{E} \approx \mathbf{P}\mathbf{Q}^T \tag{12}$$

  where $\mathbf{E}$ is the matrix of attention scores, and $\mathbf{P}$ and $\mathbf{Q}$ are low-rank matrices. This approximation reduces the computational complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \cdot r)$, where r is the rank of the approximation. This idea comes from works such as LoRA and QLoRA which are typically used in LLMs but could be applied in GNNs as well.

By combining these strategies, we could significantly optimize the self-attention mechanisms in our model, reducing computational overhead and improving overall efficiency, which is critical for cost optimization, reduced latency and improved user experience.

Exploring and Combining Other Anomaly Detection Techniques: While our focus has been on graph neural networks, there are other anomaly detection techniques that could be integrated or compared with our approach. For instance, hybrid models that combine GNNs with traditional statistical methods or machine learning algorithms could offer complementary strengths and further enhance detection performance. One possible integration is the use of an ensemble method:

$$y = \frac{1}{M} \sum_{m=1}^{M} h_m(x) \tag{13}$$

where $h_m(x)$ represents the prediction of the m-th model in the ensemble, and M is the total number of models.

Self-Supervision: In our work we have explored different self-supervised methods in which we did not obtain as good performance as in the supervised methods. For this reason, we think there is room for improvement on our methods and further work could explore variations of them or new methods. It is important as in anomaly detection datasets, it can be difficult to obtain completely labeled data.

Expanding to More Domains: Our current work has demonstrated effectiveness in the item review domain, but there are many more areas where our approach could be applied. Future research should explore additional domains, such as healthcare, cybersecurity, and IoT networks, to further validate and extend the applicability of our model. In conclusion,

while our proposed model has shown significant promise in improving anomaly detection in graph-based data in a less data-hungry and time-consuming way, there are numerous opportunities for future work to enhance its scalability, efficiency, and applicability across various domains. Addressing these challenges will help in realizing the full potential of our approach and its impact on enhancing data integrity and trustworthiness across different applications.

8 Conclusions

Firstly, taking a look at the goals we set out to achieve in the first place, we can say that we have successfully achieved each of our goals. Even though there is room for improvement on future work, we were able to develop a model capable of reliably detecting anomalies in real-time between others.

The project's impact extends far beyond our initial expectations. As we explored the model's capabilities, we saw its potential across other domains, as previously discussed. This versatility establishes our model as a powerful tool for safeguarding data integrity, reliability, and consumer protection in various sectors. Building on this success, we're confident that future iterations can further enhance the model's performance and broaden its applicability.

Overall, this project not only met its original objectives but also revealed new opportunities for further exploration and applications. It has been a journey of learning and discovery, and we're excited about the future possibilities.

References

Kipf, T. N., Welling, M. *Semi-Supervised Classification with Graph Convolutional Networks.* arXiv preprint arXiv:1609.02907, 2016.

Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A., Kaiser L., Polosukhin I. *Attention Is All You Need.* arXiv preprint arXiv:1706.03762, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y. *Graph Attention Networks.* arXiv preprint arXiv:1710.10903, 2017.

Xu, K., Hu, W., Leskovec, J., Jegelka, S. *How Powerful are Graph Neural Networks?* arXiv preprint arXiv:1810.00826, 2018.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., Veličković, P. *Principal Neighbourhood Aggregation for Graph Nets.* arXiv preprint arXiv:2004.05718, 2020.

Hafidi H., Ghogho M., Ciblat P., Swami A. *GraphCL: Contrastive Self-Supervised Learning of Graph Representations.* arXiv preprint arXiv:2007.08025, 2020.

Liu, S., Yan, X., Jin, Y. *An Edge-Aware Graph Autoencoder Trained on Scale-Imbalanced Data for Traveling Salesman Problems.* arXiv preprint arXiv:2310.06543, 2023.

Xiang, S., Zhu, M., Cheng, D., Li, E., Zhao, R., Ouyang, Y., Chen, L., Zheng, Y. *Semi-supervised Credit Card Fraud Detection via Attribute-Driven Graph Representation.* Proceedings of the AAAI Conference on Artificial Intelligence, 37(12), 14557-14565, 2023. https://doi.org/10.1609/aaai.v37i12.26702.

Xu, X., Lyu, L., Dong, Y., Lu, Y., Wang, W., Jin, H. *SplitGNN: Splitting GNN for node classification with heterogeneous attention.* Proceedings of ACM Conference (Conference'17), ACM, 2022. https://doi.org/10.1145/2301.12885.

Wang, J., Yang, G., Chen, W., Yi, H., Wu, X., Lao, Q. *MLAE: Masked LoRA Experts for Parameter-Efficient Fine-Tuning.* arXiv preprint arXiv:2405.18897, 2024.

Dettmers, T., Pagnoni, A., Holtzman, A., Zettlemoyer, L. *QLoRA: Efficient Finetuning of Quantized LLMs.* arXiv preprint arXiv:2305.14314, 2023.