A Heterogeneous Graph-based Framework for Scalable Fraud Detection

Phanindra Reddy Madduru*
maddurup@amazon.com
Amazon Inc.
USA

Naveed Janvekar
njjanvek@amazon.com
Amazon Inc.
USA

Abstract

The rise of online marketplaces has led to increased concerns regarding the presence of bad actors involved in counterfeit or engage in fraudulent activities. While efforts are being made by organizations to monitor and address these issues, bad actors persistently find new ways to engage in fraudulent behavior, including creating new accounts using different credentials, account hijacking etc. To combat this issue, our study proposes the use of Heterogeneous Relational Graph Convolutional Networks (HRGCN) to identify risky relationships among entities like sellers or customers. By leveraging this advanced graph-based approach, we aim to enhance the detection and mitigation of fraudulent behavior on the e-commerce marketplaces. The HRGCN model is designed to detect sellers with risky associations with other known bad sellers by analyzing various connecting edges such as encrypted device and identity credentials. With the rapid growth of e-commerce stores, the number of sellers has witnessed an exponential increase, leading to a significant expansion in their social networks formed by sharing various relationships such as digital contact information, communication channels and devices. This has made it challenging to process the data with the direct implementation of HRGCN. This highlights the importance of model scalability in handling large datasets. To address this issue, we have introduced a novel mini-batch version of HRGCN variant that works in tandem with a neighborhood sampler, which is optimized to run on GPUs, significantly reducing the training time by 70%. This mini-batch version of HRGCN maintains and/or improves the performance of the model while addressing the scalability issue, making it an efficient solution for handling large datasets. In this paper, we compare the performance of three models: a benchmark model based on Random Forest trained on seller node features alone, HRGCN trained on Full batch, and HRGCN with mini-batch implementation. The findings of our experiments reveal that the HRGCN models outperform the benchmark model with a significant improvement in both F1-score and Recall. Specifically, the HRGCN models show an impressive increase in recall by approximately 115% compared to the baseline model. Moreover, the mini-batch HRGCN model demonstrated substantial improvement in performance over the full batch HRGCN model, achieving a 16% higher F1 score and an 8% higher PR AUC score. These results emphasize the effectiveness of using a mini-batch approach to handle large datasets and detecting related bad sellers.

Keywords: Graph Neural Networks, Heterogeneous Relational Graph Convolution Networks, Neural Networks, Fraud Detection, Mini-batch, Fraudulent behavior, Risky relationships, E-commerce marketplaces, Model Scalability, Large datasets

ACM Reference Format:
Phanindra Reddy Madduru* and Naveed Janvekar. 2023. A Heterogeneous Graph-based Framework for Scalable Fraud Detection. In *Proceedings of 19th International Workshop on Mining and Learning with Graphs (MLG'23).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

MLG'23, Aug 2023, Long Beach, CA, USA
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

With the rapid growth of e-commerce services there are instances of scam, fraud and abuse that not only result in monetary losses for organizations but can also lead to negative customer experience. To prevent such losses and maintain customer trust, it is necessary to control fraudulent activities. Despite several programs, rule-based approaches and traditional ML models that have been developed to detect and add friction to bad actors committing fraud, these approaches are not fast enough to adapt to the rapidly evolving range of malicious behaviors. Moreover, bad actors, when enforced, constantly seek out new ways to exploit opportunities to commit fraud, such as opening new accounts with different credit cards or fake identity documents, and it is essential to detect them as early as possible to prevent any negative downstream customer impact. Therefore, identifying related bad sellers is crucial to give customers a better shopping experience.

In response, we aim to detect seller accounts that are related to known bad sellers. If a new seller shares signals such as email, phone number etc., with other known bad sellers, they can potentially be the same entity and can be considered at risk of committing fraud and abuse. However, in real-world scenarios, bad actors can have multiple virtual phone numbers and emails, making it sub-optimal to detect them using traditional methods such as tree-based classifiers [5]. To address this issue, we need more powerful graph techniques that can effectively and robustly utilize the relationships between sellers and other transaction or interaction signals.

A previous version utilizing full-batch HRGCN was successfully implemented on a smaller graph, specifically a subset of seller nodes within a designated geo-location. The smaller graph comprised approximately 3.5 million nodes, including various entities such as sellers, financial data nodes, and device nodes, resulting in over 10 million interconnected edges. However, in this study, we decided to extend our approach to include all active sellers on a global scale and consider all interconnected signals/edges to effectively assess risk across the entire graph. As a result, the graph has expanded significantly, with the number of nodes increasing by a factor of 5.5x and the number of edges increasing by a factor of 7x. Thomas and Welling (2017) have highlighted the limitation of full-batch gradient descent, where memory requirement grows linearly in the size of the dataset. They have shown that for large graphs that do not fit in GPU memory, training on CPU can still be a viable option [9]. However, through the experiments we conducted, we realized that working with large graphs on CPU is computationally expensive and time-consuming. Mini-batch stochastic gradient descent can alleviate this issue [9] which can address the limitations to make the training process more time-efficient and cost-effective. Therefore, this paper proposes an approach that utilizes graph techniques and mini-batch HRGCN training to detect bad entities in a timely and effective manner.

2 Related Work

In recent years, graph neural networks (GNNs) have emerged as a powerful framework for learning representations of structured data, such as graphs and networks. Among various types of GNNs, Relations graph neural networks (Relational GCNs) have shown to be particularly effective in modeling relationships between entities in a graph. Relational GCNs capture the structural and semantic information of a graph by aggregating information from multiple types of relations between nodes. This approach has been used for various tasks such as link prediction, recommendation systems, and fraud detection (Schlichtkrull et al., 2018; Wang et al., 2019; Yang et al., 2020).

However, the scalability of GNNs to large-scale graphs has been a major challenge. Training GNNs on a full-batch of a large graph can be computationally expensive and requires a significant amount of memory, which limits the size of graphs that can be processed. To address this limitation, several mini-batch variants of GNNs have been proposed (Hamilton et al., 2017 [7]; Chen et al., 2018 [3]). These methods randomly sample a subset of nodes and edges from the graph to form mini-batches and perform stochastic gradient descent (SGD) updates based on the sampled subgraphs.

While mini-batch GNNs have shown promising results in handling large graphs, they also face several challenges. One major challenge is the choice of mini-batch size. A small mini-batch size may lead to high variance in gradient estimation and slow convergence, while a large mini-batch size may lead to poor generalization and slow training due to the excessive computation and memory required. Another challenge is the need to store and update the subgraphs and their corresponding embeddings efficiently during training. In addition, the mini-batch sampling strategy needs to take into account the connectivity and heterogeneity of the graph to ensure that the sampled subgraphs are representative and informative (Chen et al., 2018). To overcome these challenges, recent works have proposed various techniques for mini-batch GNNs, such as adaptive mini-batch size scheduling (Chen et al., 2018), efficient subgraph embedding update (Dai et al., 2018 [4]), and neighbor sampling for scalability (Hamilton et al., 2017). Moreover, some works have explored the use of hardware accelerators, such as GPUs and TPUs, to speed up the training of mini-batch GNNs on large graphs (Sun et al., 2019).

In summary, the proposed approach builds upon the existing work in relation graph neural networks and addresses the challenges of training GNNs on large graphs by using a subgraph sampling strategy that takes into account the connectivity and heterogeneity of the graph. Our approach combines the advantages of mini-batch GNNs and Relational GCNs to effectively and efficiently detect and block bad actors in a large-scale graph of sellers and their relationships.

3 Dataset Construction

3.1 Data Sources

Seller Node Features - A seller can have businesses in multiple geographic locations with varying demographic, behavioral and transactional characteristics. To represent a seller as a single entity in our global-level model, we chose to average all the features to maintain the representation of the seller as a single node. We selected features using domain expertise, as well as the significant features from other machine learning models that aid in identifying fraudulent behavior. We processed all the above features through a Random forest model to come up with top 50 features, which are used in our analysis. To select appropriate feature values, we accessed the historical seller features data. For a bad actor seller, we chose the features closest to their account suspension date, while for a normal seller, we selected features based on the latest available data. The focus of this study is to classify seller nodes as either high-risk or low-risk based on their relatedness to other bad Sellers.

Attribute/Edge Data - This dataset comprises a wide range of historical seller signals that were used to construct the seller and attribute relationships in our graph model. For this analysis, we have handpicked a set of signals based on domain expertise. These signals encompass various attributes that have been encrypted to ensure seller privacy, such as encrypted device, identity and financial signals. Extensive measures have been taken to protect the confidentiality of seller information throughout the study. To maintain anonymity and safeguard sensitive data, specific details regarding signals used and the time period covered by the signals have been intentionally withheld.

Seller Labels - In our analysis, we define positive samples as all the sellers who's account has been suspended for fraud and abuse reasons.

3.2 Graph Construction

In order to construct our heterogeneous graph, we utilized the attribute data which includes the seller signals to establish the edges between nodes. These edges represent the relationship between the seller and the attribute signal (see Appendix 4 for sample graph structure). The node features for all seller nodes were derived from a seller feature store. The attribute nodes in the graph were initialized with random embeddings, which were refined during the optimization step [1]. During our analysis, we observed that there were some sellers with attribute data from seller relations data store but no node features available in seller feature store due to their inactivity, and vice versa. To ensure consistency in our study, we only included sellers with both attribute signals and node features. However, as a future work, we plan to expand the model to include all sellers, even those without node features, by initializing them with random, mean, or zero embedding vectors.

4 Methods

This section outlines the methods used in this study for predicting risky sellers in a heterogeneous multi-graph setup.

4.1 Random Forest

Random Forest is a widely used ensemble learning method for classification tasks. In this study, we used the Random Forest model as a benchmark to compare the performance of our proposed method. The Random Forest model was trained on the seller node features and used to predict the risky sellers.

4.2 HRGCN architecture

We employed the Heterogeneous Relational Graph Convolutional Network (HRGCN) model proposed by Schlichtkrull et al. [6] to perform classification on our heterogeneous multi-graph dataset. The HRGCN model uses the following equation for updating node representations:

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in R}\sum_{j\in N_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right) \qquad (1)$$

where $h_i^{(l)}$ is the hidden state of node i in layer l, $\mathcal{R}$ is the set of relation types, N_i^r is the set of neighbors of node i with relation type r, $c_{i,r}$ is a normalization constant, $W_r^{(l)}$ is the weight matrix for relation type r in layer l, $W_0^{(l)}$ is the weight matrix for the self-loop in layer l, and σ is the activation function (e.g. ReLU).

4.3 Mini Batch HRGCN architecture

In this section, we describe the mini-batch HRGCN algorithm implementation. We use a modified version of the HRGCN model proposed in [6] that is designed to handle large-scale graphs as it avoids the need to load the entire dataset into memory, which can be time-consuming and computationally expensive. The proposed model uses a mini-batch approach to perform training and inference efficiently.

The mini-batch HRGCN algorithm processes the graph in batches of nodes and edges. Each batch is sampled randomly from the entire graph. The model consists of several layers of HRGCNs, each of which learns a different level of abstraction. The output of the final layer is used to predict the labels of the nodes in the graph. The algorithm updates the weights of the model using Adam optimization algorithm with backpropagation. The loss function used is focus loss, which is discussed in section 4.5.

The mini-batch HRGCN model consists of several layers of HRGCNs. Each HRGCN layer takes as input a batch of nodes and edges and performs the above indicated operation (1). The input to the first layer is the feature matrix of the nodes in the batch. The output of the final layer is used to predict the labels of the nodes in the batch.

For the mini-batch HRGCN algorithm, we start by randomly selecting a batch of seller nodes B_t from the entire graph G in each iteration t. We then gather the neighbor nodes of B_t as the union of the sets of neighbors for all nodes in B_t. Denote the neighbor set of B_t as $N(B_t)$. We then construct the mini-batch subgraph G_t by extracting the nodes and edges corresponding to $B_t \cup N(B_t)$ from the original graph G. The set of edges in G_t is denoted as E_t. We then use this mini-batch subgraph G_t to perform the HRGCN algorithm.

For each layer l in the HRGCN, we first calculate the hidden representations of the nodes in the mini-batch B_t using the previous layer's hidden representations as follows:

$$h_{B_t}^{(l)} = \sigma\left(\sum_{r\in R}\sum_{j\in N_r}\frac{1}{c_{i,r}}W_{r,ij}^{(l-1)}h_j^{(l-1)} + W_{0,i}^{(l-1)}h_i^{(l-1)}\right) \qquad (2)$$

where $h_i^{(l)}$ is the hidden representation of node i at layer l, σ is the activation function, R is the set of relation types, N_r is the set of nodes that have relation r with i, $c_{i,r}$ is a normalization factor, and $W_{r,ij}^{(l-1)}$ and $W_{0,i}^{(l-1)}$ are the weight matrices of the model at layer $l-1$.

Next, we aggregate the hidden representations of the nodes in $N(B_t)$ by taking the average of the hidden representations of their neighbors in B_t. Specifically, for each node $j \in N(B_t)$, we compute its aggregated hidden representation as:

$$\hat{h}_j^{(l)} = \frac{1}{|B_t|} \sum_{i \in B_t} h_i^{(l)} W_{r,ij}^{(l-1)} \tag{3}$$

Finally, we concatenate the hidden representations of the nodes in B_t and the aggregated hidden representations of the nodes in $N(B_t)$, and use them as the input to the next layer of the HRGCN:

$$h_i^{(l+1)} = \sigma(W^{(l)} \cdot concat(h_i^{(l)}, \hat{h}_i^{(l)})) \tag{4}$$

where $W^{(l)}$ is the weight matrix of the model at layer l.

The algorithm terminates when the hidden representations of all nodes in B_t have been computed up to a desired number of layers.

Overall, the mini-batch HRGCN algorithm allows us to perform the HRGCN algorithm efficiently on large graphs by processing the graph in small, randomly selected batches. This reduces the computational complexity and memory requirements of the algorithm, while still maintaining its effectiveness in capturing the complex relationships between nodes in the graph.

4.4 Neighborhood Sampling

To make our HRGCN model more scalable, along with mini-batching we also used the HeteroGraphNeighborSampler [2] method, which is a variant of the GraphSAGE neighbor sampler [7] that is designed for heterogeneous graphs. The HeteroGraphNeighborSampler takes as input a graph $G = (V, E)$, where V is the set of nodes, E is the set of edges. Given a set of starting nodes $S \subseteq V$, the HeteroGraphNeighborSampler returns a set of subgraphs that consist of k hops from the starting nodes, with k being a hyperparameter. Specifically, it samples the immediate neighbors of the starting nodes in the first hop, and then iteratively samples the neighbors of the previously sampled nodes in the subsequent hops, until k hops are reached. This results in a set of "neighborhoods" for each node in the batch. We then use these neighborhoods to construct a subgraph of the original heterogeneous graph. This subgraph contains only the nodes and edges that are relevant to the nodes in the batch. We use this subgraph to perform message passing and update the node embeddings.

By using the HeteroGraphNeighborSampler method, we are able to perform mini-batch training on our HRGCN model. This significantly reduces the training time and allows us to train on larger datasets. In addition, the use of neighborhood sampling allows us to capture the local structure of the graph. This is important for heterogeneous graphs, where nodes of different types may have different connectivity patterns. Overall, the HeteroGraphNeighborSampler method is a powerful sampling technique for performing mini-batch training on heterogeneous graphs. Its use in our HRGCN model allows us to achieve better scalability and performance compared to the full-batch approach.

4.5 Focal Loss Function

We use the Focal Loss function instead of a cross-entropy loss function to address the class imbalance problem in our dataset. The key insight of the Focal Loss function is that it down-weights the contribution of easy examples to the loss, which in turn allows the model to focus more on hard examples during training. This is particularly useful in scenarios where the majority class is much larger than the minority class, as the model tends to overfit to the majority class otherwise [8]. In comparison to the binary loss function, the Focal Loss function can improve the performance of the model on the minority class without sacrificing performance on the majority class. The Focal Loss function achieves this by assigning a higher weight to the minority class during training, which can help the model learn more from the minority class examples. This approach can result in a more balanced model that can handle the minority class with higher accuracy, making it more suitable for real-world scenarios where detecting rare events is critical. Specifically, we observed that the number of non-risky sellers greatly outnumber the number of risky related sellers in our dataset. This can lead to a bias towards the majority class during training, which can result in poor performance on the minority class [8, 10].

$$L_{FL}(p_t) = -\alpha_t (1 - p_t)^{\gamma} \log(p_t) \tag{5}$$

where p_t is the predicted probability for the positive class (probability of being a bad actor), α_t is a balancing factor that weights the contribution of each class based on their frequency, and γ is a tunable parameter that controls the degree of focus. When γ is set to 0, the focal loss reduces to the standard cross-entropy loss, while larger values of γ put more emphasis on hard examples.

5 Experimental Setup

In this section, we present the experimental setup for our study on detecting risky sellers using below 3 candidate methods. We discuss the candidate models, the data splitting approach, and the modeling hyperparameters. We evaluated the performance of the models using precision, recall, F1-score, AUC-PR and AUC-ROC. We then present the results of the experiments and compare the performance of each model. Finally, we discuss our observations and draw conclusions based on the findings.

5.1 Candidate methods

To ensure a fair comparison, we used the same training and test datasets for all the models. To rule out effects of overfitting, we performed training in 10-fold CV with a left-alone set of sellers only for test purposes for all the methods.

1) RF. Its a Random Forest classification model with 100 estimators and maximum depth of 10. This model uses the selected top 50 seller features to predict the riskiness of a seller.

2) HRGCN Full-Batch (*$HRGCN_{Full}$*). Its the HRGCN implementation with 50 features assigned to each seller node, using a 5-layer NN with ReLU actiation between the layers and varying hidden dimensions (50 until l - 2 layers, 16 for l - 1 layer, and 2 for the output layer). The model is trained for 200 epochs with an early stopping criterion based on validation loss. We use focal loss with Adam optimizer (learning rate=0.003) and L2 regularization to the weights with a weight decay of 0.0003 to minimize the loss function. Dropout is applied ahead of each hidden layer, but did not significantly improve performance, hence it is set to 0. The hyperparameters are chosen via grid search, and the best performing parameters from training the model with the full batch HRGCN are used for the mini-batch implementation as well to ensure a fair comparison. The model is trained on CPU due to the inability to process the full graph on GPU.

3) HRGCN Mini-Batch (*$HRGCN_{Mini}$*). To improve the scalability of the HRGCN model, we also implemented a mini-batch training approach with neighbor sampling. Specifically, we randomly selected 10 edges for each node and used a batch size of 2^{14} for training. We set the learning rate to 0.0001 and trained the model for 15 epochs. To ensure efficient performance, we trained the mini-batch model on a GPU. The rest of the implementation was the same as the full-batch HRGCN, with a 5-layer neural network using ReLU activation, focal loss with Adam optimizer and L2 regularization with a weight decay of 0.0003. All the updated hyperparameters for this implementation were also chosen via grid search, and rest are set based of hyperparameters that achieved the best performance in the full-batch HRGCN. Algorithm 1 is the high-level pseudo code implementation of the Mini-Batch HRGCN that was carried out in this study.

Algorithm 1: Mini-Batch HRGCN with Focal Loss and HeteroGraphNeighborSampler

```
Input: Heterogeneous graph $G = (V, E)$, with $N$
       nodes and $K$ node types, and node features $X$
Output: Model parameters $\theta$
Initialize model parameters $\theta$
Initialize mini-batch size $B$, number of epochs $E$,
  learning rate $\alpha$, and $\gamma$ for Focal Loss
Initialize number of layers $l$ and
  HeteroGraphNeighborSampler with sampling
  strategy
for epoch in $1...E$ do
    for mini-batch $b$ in $1...N/B$ do
        Sample $B$ nodes from each node type using
          HeteroGraphNeighborSampler
        Construct a subgraph $G_b$ containing the
          sampled nodes and their neighbors
        Compute embeddings for nodes in $G_b$ using
          R-GCN
        Compute the probability distribution over
          classes for each node using a fully-connected
          layer and softmax activation
        Compute Focal Loss using the probability
          distribution and node labels
        Update model parameters $\theta$ using
          backpropagation and the Adam optimizer
          with learning rate $\alpha$
    end
end
```

5.2 Training and Evaluation Results

The dataset contains over 4.3 million sellers of which a portion of them being fraudulent. We randomly divide these sellers into 11 even sets, leaving one set out for testing, and the other 10 acting as the non-overlapping validation set. In each fold, nearly 400K non overlapping sellers are set as validation sellers maintaining the same fraud event ratio. The fraud event ratio is intentionally withheld to safeguard sensitive data. The dataset contains 13.7M attributes shared by all the sellers in the whole graph creating 61M edges. For training purpose, we include all the sellers and attribute nodes into the graph but the validation sellers will be considered as unlabeled sellers. Which means, the model only uses the labeled nodes to compute the loss function and update the parameters. The unlabelled nodes are used to compute the message passing operation, but their representations are not used in the loss computation.

From Fig 1, presents the results of the experiment on two variations of the HRGCN model, namely *$HRGCN_{Mini}$* and *$HRGCN_{Full}$*, both of which were run for a different number of epochs - 16 and 200, respectively. However, for real-world scenarios, we implemented early stopping, wherein the training was stopped if the F1 score or loss didn't improve for more than 10 epochs. The figure also includes the F1 score of the Random Forest model as a baseline reference.The results indicate that the F1 score of *$HRGCN_{Mini}$* converges and reaches its peak performance by the 9th-10th epoch, whereas *$HRGCN_{Full}$* converges around the 180-200th epoch. The validation loss of *$HRGCN_{Mini}$* drops rapidly within the first five epochs and then stabilizes, indicating that the model converges quickly. Interestingly, *$HRGCN_{Mini}$* reaches an F1 score of 0.77, which is the average F1 score, in 180 minutes, whereas *$HRGCN_{Full}$* only reaches 0.64 in the same

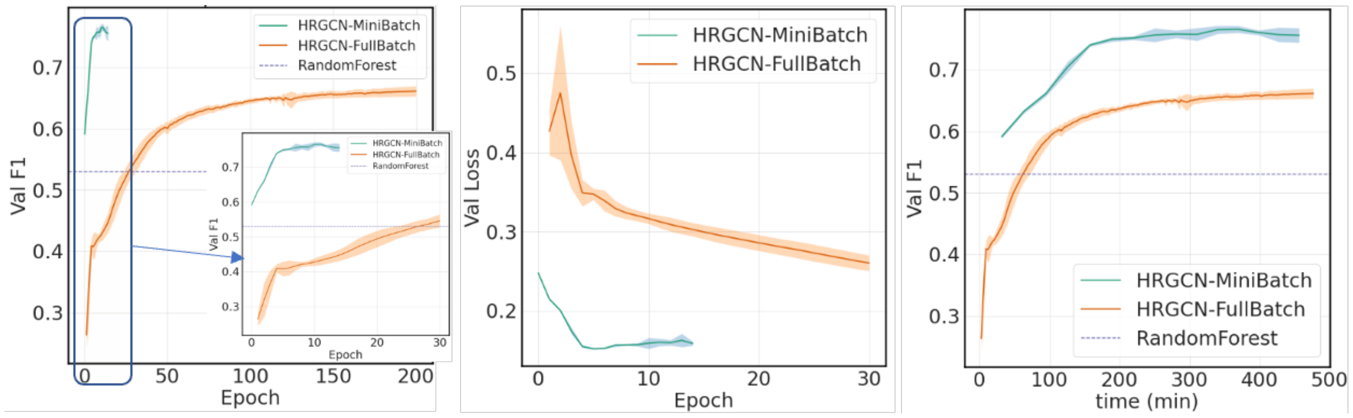

Figure 1. F1 and Loss metrics of different models on Validation Set with respect to Epoch and Time.

Model	Accuracy	F1 Score	Precision	Recall	ROC AUC	PR AUC
HRGCN-Mini	0.913±0.003∗	0.772±0.003∗	0.735±0.019	0.814±0.017	0.941±0.002∗	0.849±0.004∗
HRGCN-Full	0.848±0.005	0.666±0.004	0.555±0.012	0.835±0.018∗	0.92±0.005	0.784±0.014
RandomForest	0.878±0.001	0.531±0.003	0.883±0.002∗	0.38±0.003	0.89±0.001	0.737±0.002

Table 1. Aggregated Results for Each Model

time. $HRGCN_{Full}$ takes around 450-500 minutes (8 hours) to converge, whereas $HRGCN_{Mini}$ takes only 180 minutes (3 hours). The most interesting observation is the time taken for the model to converge. The $HRGCN_{Mini}$ reaches to 0.77 F1 score (which is the average F1 score for $HRGCN_{Mini}$) in 180 min where as the $HRGCN_{Full}$ reaches only 0.64 in the same time. The $HRGCN_{Full}$ converges around 450-500 min (8 Hours) while the $HRGCN_{Mini}$ converges around 180 min (3 hours). During the experiments, we noticed a deceleration in the F1 score and validation loss of $HRGCN_{Mini}$ after the 12th epoch, despite the training dataset metrics showing an improvement. This behavior could be indicative of overfitting. In order to mitigate this issue, we implemented early stopping in the training of production models, halting the training process by the 10th epoch, thereby improving the efficiency of the training process as well as time taken to train.

Table 1 shows the performance of the both the HRGCN models and Random Forest models on the validation set. The $HRGCN_{Mini}$ model has the highest scores in all metrics, except Precision where it is slightly lower than the *RandomForest* model. The $HRGCN_{Full}$ model has lower scores compared to the $HRGCN_{Mini}$ model, but it still performs better than the RandomForest model in most metrics.

These results indicate that the $HRGCN_{Mini}$ model outperforms both the $HRGCN_{Full}$ and RandomForest models in most metrics. The $HRGCN_{Mini}$ model also has the highest ROC AUC and PR AUC scores, indicating that it performs better at identifying risky related sellers compared to the

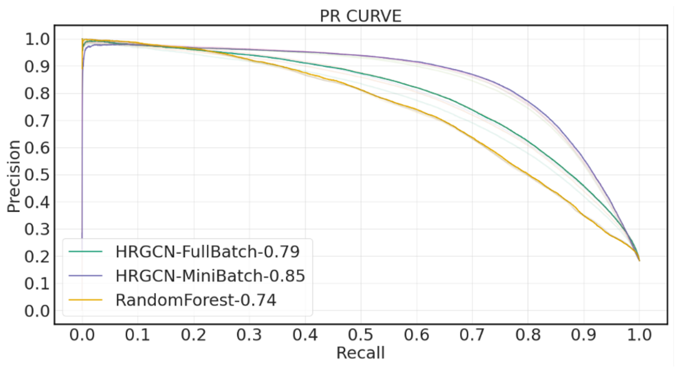

Figure 2. PR Curves Comparing the both HRGCN models' and baseline model RandomForest

other models. The results highlight the significance of fine-tuning the HRGCN model's architecture to attain optimal performance and scalability, particularly for handling extensive datasets.

5.3 Seller clustering using learned embeddings

To assess the performance of the HRGCN models' learned embeddings, we plotted the embeddings using the t-distributed stochastic neighbor embedding (t-SNE) technique. The embeddings were colored by their ground truth labels to help distinguish the classes. From the resulting plot, we observed that both models produced separable clusters of classes, indicating that they have learned meaningful representations of the graph data. However, there was a clear difference in the quality of the embeddings between the two models.

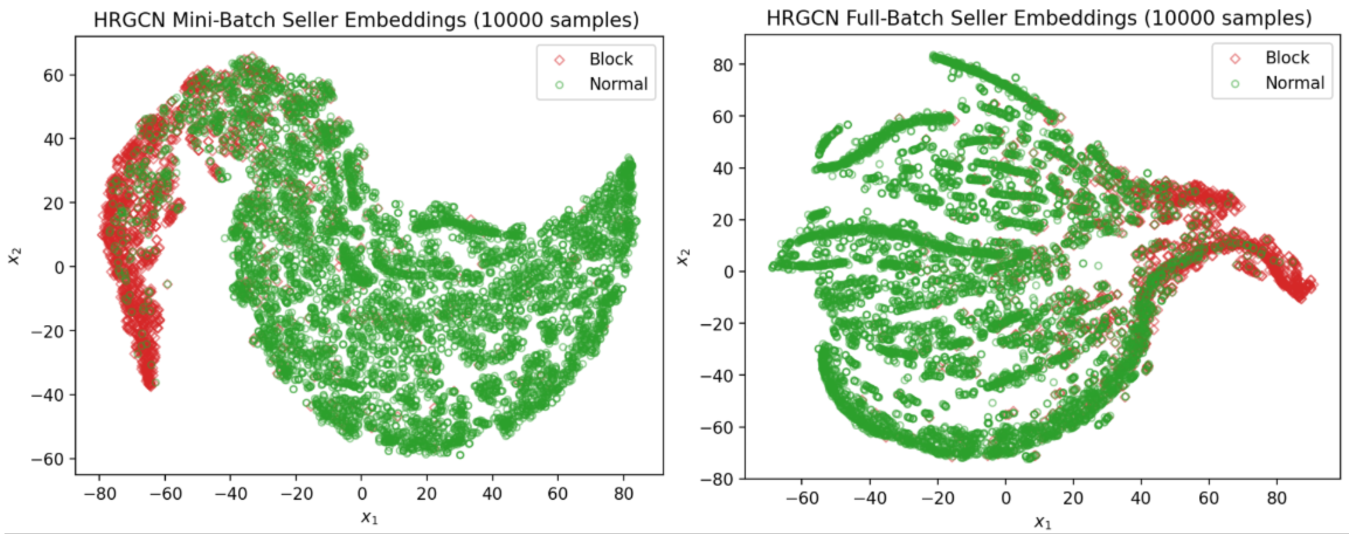

Figure 3. t-SNE Plot of the Seller Embeddings Generated by both the HRGCN models

Specifically, we observed that the embeddings generated by $HRGCN_{Mini}$ were much better separated and formed more distinct clusters compared to $HRGCN_{Full}$. The clusters in $HRGCN_{Mini}$ were more tightly packed and had less overlap between classes, indicating a higher level of accuracy in the model's classification performance. In contrast, $HRGCN_{Full}$ produced more spread-out clusters with more overlap between classes, indicating a lower level of accuracy in the model's classification performance. Therefore, we can conclude that the embeddings generated by $HRGCN_{Mini}$ are more accurate and useful for clustering the data.

6 Conclusion and Future Work

In this study, we investigated the HRGCN architecture and its scalability for predicting risky related sellers on large datasets. We introduced a new variant of HRGCN, $HRGCN_{Mini}$, and compared its performance with the traditional $HRGCN_{Full}$ implementation on detecting bad actors. Our results indicate that the $HRGCN_{Mini}$ model outperforms the $HRGCN_{Full}$ model in terms of the F1 score, validation loss, and convergence time. Specifically, $HRGCN_{Mini}$ achieved its maximum performance in 2.5 hours, while $HRGCN_{Full}$ took 8 hours to converge, reducing training time by almost 70%. Moreover, the precision recall metrics show that $HRGCN_{Mini}$ outperforms $HRGCN_{Full}$ and Random Forest models, achieving a recall of 63% at 90% precision (see Figure 2 for PR-Curve comparisons). This represents an 80% improvement over the baseline model and a 46% improvement over the $HRGCN_{Full}$ model. The embeddings generated by $HRGCN_{Mini}$ also demonstrate better class separability compared to $HRGCN_{Full}$. Our study emphasizes the importance of optimizing the HRGCN architecture to achieve high performance and scalability for large datasets. The outcomes of our research can offer important insights for researchers working on similar problems. In our forthcoming work, we aim to investigate supplementary methods to enhance the performance of HRGCN. Additionally, we intend to incorporate the HRGCN Explainer component to create a comprehensive framework. Additionally, we aim to leverage the embeddings generated by HRGCN for downstream tasks. One such application is to transition from individual seller analysis to identifying clusters of bad actors. This approach will require applying clustering techniques such as K-Means on the generated embeddings to identify clusters of bad actors that are linked through various relationship edges.

References

[1] Jaime Acevedo-Viloria, Luisa Roa, Soji Adeshina, Cesar Olazo, Andrés Rodríguez-Rey, Jose Ramos, and Alejandro Correa-Bahnsen. 2021. Relational Graph Neural Networks for Fraud Detection in a Super-App Environment. *ArXiv preprint arXiv:2106.04341* (2021).

[2] awslabs. [n. d.]. Implementation of HeteroGraphNeighborSampler. ([n. d.]). https://github.com/awslabs/sagemaker-graph-fraud-detection/blob/master/source/sagemaker/sagemaker_graph_fraud_detection/dgl_fraud_detection/sampler.py.

[3] H. Chen, E. Kiciman, W. Marczak, and Y. Cheng. 2018. GCMC: Graph Convolutional Matrix Completion. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[4] H. Dai, Z. Kozareva, B. Dai, A. Smola, and L. Song. 2018. Learning Steady-States of Iterative Algorithms over Graphs. In *Proceedings of the 35th International Conference on Machine Learning*.

[5] Boning Li et al. 2022. GNN-based Seller Risky Relations Model. In *Amazon Machine Learning Conference*.

[6] Michael Schlichtkrull et al. 2018. Modeling Relational Data with Graph Convolutional Networks. In *European Semantic Web Conference*. Springer, 593–607.

[7] W. L. Hamilton, Z. Ying, and J. Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[8] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. 2017. Focal Loss for Dense Object Detection. In *Proceedings of the IEEE International*

Conference on Computer Vision. 2980–2988.

[9] N. Kip Thomas and Max Welling. 2017. Semi-supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

[10] T. Zhou, Z. Jia, Q. Li, and J. Li. 2021. Research on the Application of Focal Loss in the Field of Imbalanced Data. *Multimedia Tools and Applications* 80, 10 (2021), 15029–15044.

7 Appendix

A Sample Network Structure

A sample sub-graph of a seller node connected to various signals and other known bad actors.

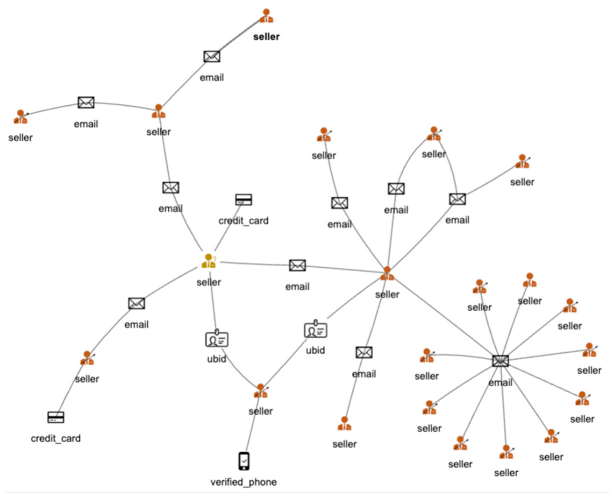

Figure 4. An example of the graph structure is presented, illustrating how sellers are interconnected through various attributes such as email, ubid, and financial signals.