xFraud: Explainable Fraud Transaction Detection

Susie Xi Rao†, Shuai Zhang†, Zhichao Han★, Zitao Zhang★, Wei Min★, Zhiyao Chen★, Yinan Shan★, Yang Zhao★, Ce Zhang†
{raox,shuazhang,ce.zhang}@inf.ethz.ch
{zhihan,zitzhang,wmin,zhiyachen,yshan,yzhao5}@ebay.com
ETH Zurich† eBay China★

ABSTRACT

At online retail platforms, it is crucial to actively detect the risks of transactions to improve customer experience and minimize financial loss. In this work, we propose xFraud, an explainable fraud transaction prediction framework which is mainly composed of a detector and an explainer. The xFraud detector can effectively and efficiently predict the legitimacy of incoming transactions. Specifically, it utilizes a heterogeneous graph neural network to learn expressive representations from the informative heterogeneously typed entities in the transaction logs. The explainer in xFraud can generate meaningful and human-understandable explanations from graphs to facilitate further processes in the business unit. In our experiments with xFraud on real transaction networks with up to 1.1 billion nodes and 3.7 billion edges, xFraud is able to outperform various baseline models in many evaluation metrics while remaining scalable in distributed settings. In addition, we show that xFraud explainer can generate reasonable explanations to significantly assist the business analysis via both quantitative and qualitative evaluations.

PVLDB Reference Format:
Susie Xi Rao†, Shuai Zhang†, Zhichao Han★, Zitao Zhang★, Wei Min★, Zhiyao Chen★, Yinan Shan★, Yang Zhao★, Ce Zhang†. xFraud: Explainable Fraud Transaction Detection. PVLDB, 15(3): XXX-XXX, 2022.
doi:10.14778/3494124.3494128

PVLDB Artifact Availability:
Appendix, code and sample data under https://github.com/eBay/xFraud

1 INTRODUCTION

The online retail industry is reshaping our shopping behavior, and the resulting security risks are not negligible. Common threats in e-commerce include account acquisition, financial information theft, fake chargeback, money laundry, and many more. For instance, malicious attackers might try to steal customer's credit card information; the login credentials can also be acquired by hackers. These criminal activities can bring negative impacts on user experiences, cause financial losses, and seriously degrade the platform credibility. As such, it is critical to identify fraudulent behaviors and take every precaution to minimize risks.

Proceedings of the VLDB Endowment, Vol. 15, No. 3 ISSN 2150-8097.
doi:10.14778/3494124.3494128

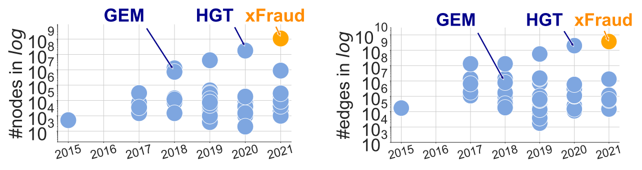


Figure 1: The node and edge numbers (*log*) of heterogeneous graph datasets in the literature. Full survey in Appendix A [34].

Fraud detection has been an emerging topic for e-commerce and social media companies. It is studied in various applications: malicious account detection (e.g., social networks [3], online payment systems [53], and online retailer platforms [4, 5, 28, 54]); anti-money laundry [10, 43]; spam reviews and news detection [38, 41]. Among these *transaction fraud detection* is an important topic [4, 5, 54]. In this work, we focus on automatic fraudulent transaction detection in a real-world e-commerce environment at eBay. For an incoming transaction, we aim to predict whether it is legitimate or not.

Challenges. Despite recent efforts in automatic fraudulent transaction detection [3, 8, 11, 17, 20, 22, 26, 28, 33, 38, 40, 44, 51, 53] with machine learning such as LSTM and graph neural networks (GNNs), we realize that three challenges still linger when it comes to our application scenario at eBay. ***(1. Information Heterogeneity)*** In our system, there are heterogeneous types of information concerning a transaction such as payment tokens, shipping addresses, email. Intuitively, such information is indicative in fraud detection. *How to effectively utilize such information in an end-to-end ML model?* ***(2. Scalability and Efficiency)*** Our platform can produce millions of transactions involving millions of users in a short span of time, which requires the detecting system to be *efficient and scalable for practical use.* Specifically, Figure 1 shows the landscape of heterogeneous graphs emerging in the last six years. In this paper, we are tackling a workload, to our best knowledge, that consists of one of the largest *heterogeneous* graphs for graph neural networks (see a more detailed survey in Appendix A [34]). This poses unique challenges in the system design and optimizations. ***(3. Explainability)*** Flagging a transaction to be fraudulent is not a trivial process. False decisions are likely to cause trouble to our customers and significantly degrade the platform's credibility. In general, this process requires extensive human efforts in cautiously reviewing the model's prediction, which is inefficient and costly. *How can we explain the outcome of an ML model, and more importantly, how close these explanations are to those developed by human experts in the business unit (BU)?*

Our Approach. To tackle the aforementioned challenges, we propose xFraud, an explainable fraud detection framework at eBay. xFraud is not only able to efficiently and effectively predict the legitimacy of a transaction but can also generate human readable explanations in to assist flagging transaction frauds. xFraud advances previous work in two ways.

First, xFraud builds upon a heterogeneous GNN to tackle *transaction fraud detection.* Specifically, xFraud consists of a detector and an explainer. In the detector, we tackle a transaction fraud detection task from the graph perspective. Different from the existing works [24, 27, 31], a heterogeneous graph of different node types (e.g., transactions, addresses, payment tokens) is constructed. To capture the heterogeneous relation patterns and learn more expressive node representations, a self-attentive heterogeneous graph neural network is adopted. The detector can automatically aggregate information from different types of nodes via disparate paths without manually predefined meta-paths. This is important because we do not need to predefine the meta-paths of risk propagation and preprocess the path representations, as required in [4, 17, 38, 53]. Additionally, we explore an efficient sampler in the message aggregation procedure, which empirically reduces the inference time significantly while achieves a competitive performance.

Second, to the best of our knowledge, we provide the first quantitative evaluation of input-level GNN explanation methods [45, 47] and its agreement with humans in a real-world application scenario. Unlike traditional classification tasks, the claim that a transaction is fraudulent should be made very cautiously to avoid harming customer experience and degrading the platform's credibility. As such, we integrate an explainer into our framework that can provide intuitive explanations for model predictions. Equipped with these explanations, our auditors, regulators, or decision makers know how a transaction is flagged by the detector, thus making more sensible decisions. One open question is *how well these explanations agree with the insights from human experts.* To this end, we conduct an extensive quantitative study to measure the agreement between human perception, GNNExplainer [47], and centrality measures, as well as provide case studies on how these explanations can help in practice. This study also reveals an interesting tradeoff between GNN-based explanations and traditional topological measures (e.g., centrality), which allows us to design a hybrid explainer that outperforms both strategies.

In summary, our technical contributions are as follows.

(1) We propose a heterogeneous GNN model (detector) to identify transaction frauds. Our model captures the heterogeneity in transaction graphs and can be applied to industrial-scale datasets. xFraud detector provides concrete analytical angles of fraudulent activities. Compared with HGT [18], in xFraud, we design a new sampling mechanism, inspired by the sparsity of our underlying graph; compared with GEM [28], xFraud uses a heterogeneous GNN architecture, which allows it to outperform a GEM-style model significantly.

(2) We add explainability into xFraud with a hybrid explainer. xFraud explainer computes the contributions of its neighboring node types and edges when predicting a node, and it also attends to global topological features learned from centrality measures. Thus, it enables explicit case studies of network predictions, which is beneficial for model trustworthiness. We perform the first quantitative evaluation between GNNExplainer and human judgments. We also compare edge weights computed via centrality measures with the weights learned by GNNExplainer, through which we identify a trade-off and propose a hybrid explainer in xFraud.

(3) We conduct careful system design and optimizations, which allow us to scale out, to our best knowledge, to one of the largest heterogeneous graphs being reported for ML workloads so far.

(4) We conduct experiments on real-world transaction networks to show the efficiency of xFraud in detecting transaction frauds and in facilitating the analysis of graph structural patterns.

2 RELATED WORK

xFraud builds upon recent successes of applying heterogeneous graphs to fraud detection (e.g., MAHINDER [53] and GEM [28] from Alibaba) and also recent efforts of GNN explainability [47]. However, it also makes significant improvements over these previous efforts, discussed as follows.

Fraud Detection. There are two lines of studies on fraud detection systems. One line of work leverages graph information with non-GNN methods (see anti-money laundry [10], spam detection [8, 20, 38], fraudster user detection [3, 11, 15, 17, 21, 23, 26, 33, 40, 53], fraud transaction detection [4, 5, 35, 54]). These models can be event/sequence based [3, 5, 22, 54], or meta-path based [4, 17, 38, 53]. Zhong et al. [53] propose MAHINDER which uses *heterogeneous graphs* in the context of defaulter detection by pre-defining a set of *meta-paths* in a heterogeneous graph of users and merchants. The preprocessed meta-path feature representations are trained with an attention mechanism and LSTM to measure the importance of nodes, links, and meta-paths at different timestamps. In xFraud, we focus on a different scenario, aiming at flagging each transaction of a user in various risk scenarios, as a legitimate user does not imply that all its transaction records are legitimate, e.g., once its payment token has been stolen. More importantly, we *focus on methods that do not need to define meta-paths a priori, instead are able to automatically learn these patterns using a GNN.*

The other line of work uses GNN methods [22, 24, 27, 28, 31, 41, 43, 44, 51]. Homogeneous graph has been widely applied in e-commerce applications (see spam review detection [41], anti-money laundry [43], risky/malicious account detection [24, 27, 31]). Recently, people start to solve real-world anomaly detection problems using heterogeneous graph (see spam review detection [22], suspicious user detection [28, 44, 51]) or to combine homogeneous and heterogeneous graphs [22], because it allows aggregating information propagation through various types of nodes/edges. GEM [28] by Liu et al. has utilized attention mechanisms in a device-account heterogeneous graph to capture user activity and device embeddings in each subgraph neighborhood. The heterogeneous graph in GEM is then fed into a GCN.

Explainability in GNN. Recently, how to interpret and explain GNN predictions has gained spotlight. There are two levels to explain: input level (GraphConsis[29], GNNExplainer [47], GraphLIME [19], [2, 45]), and model level (XGNN [48]). GNNExplainer [47], a GNN model agnostic explanation framework, proposes explaining the GNN predictions by maximizing the mutual information gain of the true node labels and the predicted labels using informative features. GNNExplainer enables a visualization of important subgraph patterns, which assists users to understand the feature contribution and node label propagation. GNN explainability in the financial domain has been addressed by Li et al. [45]. They have extended GNNExplainer techniques by (1) adding a regularization term that ensures at least one edge connected to each

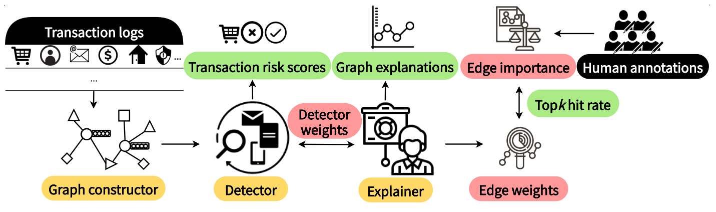


Figure 2: xFraud pipeline.

Transaction	Buyer	Email	Payment token	Shipping address	Transaction features
1	1	john_eth@gmail.com	Credit card	Albert-Einstein-Strasse 1, Zurich	...
2	1	john_eth@gmail.com	Payment slip	Hauptstrasse 1, Zurich	...

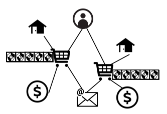


Figure 3: Transactions → a heterogeneous graph.

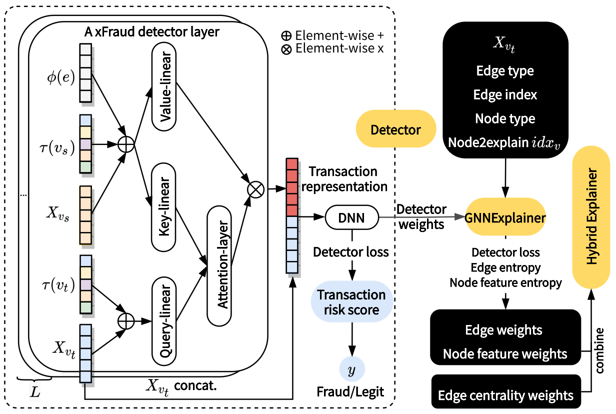


Figure 4: xFraud detector and explainer.

node is selected in the subgraph, (2) adding edge weighted graph attention to calculate the edge weights in the subgraph. Using GCN in a node classification task, they applied an explainer to identify informative graph patterns on financial transaction data like bitcoin over the counter (OTC) and account matching in bank transactions.

3 THE XFRAUD FRAMEWORK

Figure 2 illustrates the xFraud pipeline in a nutshell. First, we build a graph constructor to convert transaction logs into a graph abstraction (Sec. 3.1), which is then fed into a detector to generate a transaction risk score for each transaction record (Sec. 3.2). Then, to build a learnable hybrid explainer (Sec. 3.4), we combine the task-aware measures of predictions generated by the GNNExplainer, and the task-agnostic centrality measures.

One highlight of xFraud over previous efforts is its emphasis on explainability, especially its evaluation using insights from real-world experts in the business unit. We evaluate the efficacy of the explainer quantitatively (Sec. 5.1) and qualitatively (Sec. 5.2). Quantitatively, we calculate the agreement (top$k$ hit rate) between human annotations and explainer weights. Concretely, we first obtain human ground truth on edge importance in risk propagation. Then, we calculate edge importance scores from node importance scores with various aggregation methods. At last, we report the top$k$ hit rate between edge importance scores computed from human annotations and edge weights generated by the hybrid explainer. Qualitatively, we study in detail many cases where xFraud explainer assists the BU in better understanding complex fraudulent patterns.

3.1 Heterogeneous Graph Construction

Think of the critical entities involved under fraud scenarios. A credit card might be linked to both a legitimate user and a fraudulent user at different stages. The latter happens in a card stolen case. A common shipping address such as a warehouse is sometimes used in frauds. This linkage tends to be stable, compared with stolen financial instruments. If we formulate fraud detection as a semisupervised learning problem in an inductive setting [14] in a heterogeneous graph, we have the specification of the problem formulation as follows. In a heterogeneous transaction graph $\mathcal{G}$, $v \in \mathcal{V}$ has a type $\tau(v) \in \mathcal{A}$, where $\mathcal{A} := \{txn, pmt, email, addr, buyer\}$, referring to *transaction, payment token, email, shipping address, buyer*, respectively[1]. If a transaction has relation with another type of node in $\{pmt, email, addr, buyer\}$, we put an edge between those two nodes in the heterogeneous graph. Each txn node carries node attributes provided by a risk identification system. A transaction is represented as an ID in the transaction log. The item category in the purchase order relevant to one transaction (item-type info) is encoded in the transaction features. Each transaction is flagged legit or fraud. Figure 3 illustrates how to construct such a heterogeneous graph based on two transaction records sharing several entities.

3.2 The detector of xFraud

As we have seen in the literature, it is common to define meta-paths when analyzing graph structured data and then to extract corresponding features of nodes and edges on the meta-paths before feeding the features into a machine learning or deep learning model. However, in a fraud detection scenario, under many circumstances, it is by nature impossible to enumerate every possible scenario and their influential meta-paths. This is also one of the primary intuitions why a heterogeneous GNN is a desirable choice: it allows a network to learn the importance of meta-paths by itself based on the network structure and message passing.

3.2.1 ***xFraud detector.*** We are inspired by Transformer [39] and HGT [18], when designing the xFraud detector incl. heterogeneous mutual attention and heterogeneous message passing with key, value, and query vector operations (self-attention mechanism). We do not allow target-specific aggregation on different node types, so that we reduce the cost in computing different weights for various node types. We see a better performance in our detector (see discussion in Sec. 4), when shared weights among different types of nodes are used. Moreover, we do not adopt relative temporal encoding in HGT when processing transactions with timestamps. Reasons are, we would like to keep track of all transactions a buyer executes, as well as the linking entities a transaction involves. We also model the relations between buyers and transactions. This makes our system adaptable to guest checkouts and their pertaining chargebacks, as those transactions could not be linked to any buyer accounts, but they could be linked to suspicious third-party

[1]For this study, we choose these attributes based on the homophilic tests [1]. It is shown that fraud exhibits homophilic effects [32], and entities with strong homophilic effects are considered in this work.

payment accounts or billing email addresses. In this manner, our system is able to capture disguised/missed fraud patterns of guest checkouts that could otherwise be neglected by (1) representing a transaction using buyers and timestamps as in HGT or by (2) representing the transactions in a homogeneous graph.

Comparison to GEM. Looking at the application scenarios, xFraud might seem similar to GEM (a malicious account detection system developed by Alibaba). However, xFraud differs from GEM [28] in that: (1) GEM is a system which directly applies a vanilla GCN to a heterogeneous graph, while our proposed xFraud considers the heterogeneous property of graphs in the underlying architecture (e.g., sampler, heterogeneous graph convolution). In this paper, we choose GEM as a representative of heterogeneous GCN in the evaluation. (2) We focus on a different application and have different node types. GEM focuses on fraudsters' detection, while we aim to find the anomaly transactions (a user may have both fraudulent and normal transactions due to account hacking). (3) The GEM model does not provide any explanation for its predictions. We have extensively discussed the GNN explainability and conducted experiments to understand the xFraud explainer.

In Figure 4, we show the detector architecture in detail (left). **(1)** The detector takes a heterogeneous graph as input, incl. target and source node features X_{v_t}, X_{v_s}; target and source node types $\tau(v_t)$, $\tau(v_s)$; edge type $\phi(e)$. For the *txn* nodes, we have node features computed by a company risk identifier. For the other node types, the initial node features are empty and only get their inputs after the first convolution layer. The type features are in one-hot encoding of types. **(2)** *L heterogeneous convolution layers* process the graph with self-attention mechanism: the input layer $\mathrm{L}^{(0)}$ takes transaction features, node type embeddings (source and target), edge type embeddings as input, which are transformed into query, key, and value vectors. Attention scores are calculated for the source and target nodes and then layer-wise normalized, which are then fed into a *ReLU* activation function that emits input for the next convolution layer. In Sec. 3.2.2, we introduce with equations how heterogeneous mutual attention and message passing function in one heterogeneous convolution layer. **(3)** After L heterogeneous convolution layers, a *tanh* activation is applied to the transaction representations generated by GNN. Then these representations are concatenated with the original transaction features and fed into a feedforward connected network with two hidden layers. We then apply dropout, layer normalization, and *ReLU* transformation to calculate a predicted risk score and a label. **(4)** The loss function of xFraud detector is the cross entropy of the true label and the probability score calculated by *softmax* (see eq. 1 in Appendix D [34]).

3.2.2 *Heterogeneous Convolution Layer in xFraud Detector.*

We discuss the details of a xFraud detector layer shown in Figure 4. For a tuple $\langle \tau(v_s), \phi(e), \tau(v_t) \rangle$, where $e = (v_s, v_t)$, we initialize (1) the node type embeddings $\tau(v)^{emb}$ and the edge type embeddings $\phi(e)^{emb}$ with zero weights; (2) the attention weight matrices of source node $W^{att}_{\tau(v_s)}$ and of target node $W^{att}_{\tau(v_t)}$ with random weights subject to uniform distributions; and (3) the weight matrices for key, query, and value vectors denoted by W^K, W^Q, W^V, respectively, also with random values subject to uniform distributions. In a nutshell, a general attention-based heterogeneous convolution layer of the node v_t has three components, attention, message, and aggregate as shown in $H^l[v_t] \leftarrow Aggregate(Attention(v_s, v_t) \cdot Message(v_s))$. For each target node v_t, we create *query, key*, and *value* vector representations for self-attention mechanism with multiheads.

To construct the ith *query* vector for the target node v_t, we start with an input to the first layer by taking the transaction features of the target node $X^{txn}_{\tau(v_t)}$ and its node type embedding $\tau(v_t)^{emb}$ to calculate $Q^i(v_t) = \text{Q-Linear}^i_{\tau(v_t)}\left(X^{txn}_{\tau(v_t)} + \tau(v_t)^{emb}\right)$, Then, for $H^{(l-1)}$, where $l \neq 1$, we compute $Q^i(v_t) = \text{Q-Linear}^i_{\tau(v_t)}\left(H^{(l-1)}[v_t]\right)$, where $H^{(l-1)}[v_t]$ is the node representation of the node v_t on the $H^{(l-1)}$ layer.

To construct the ith *key* vector for the source node v_s, we start with an input to the first layer by taking the transaction features of the source node $X^{txn}_{\tau(v_s)}$, its node type embedding $\tau(v_s)^{emb}$ and the edge type embedding $\phi(e)^{emb}$ to calculate $K^i(v_s) = \text{K-Linear}^i_{\tau(v_s)}\left(X^{txn}_{\tau(v_s)} + \tau(v_s)^{emb} + \phi(e)^{emb}\right)$, Then, for $H^{(l-1)}$, where $l \neq 1$, we compute $K^i(v_s) = \text{K-Linear}^i_{\tau(v_s)}\left(H^{(l-1)}[v_s]\right)$, where $H^{(l-1)}[s]$ is the node representation of the node v_s on the $H^{(l-1)}$ layer.

To construct the ith *value* vector for source node v_s, we start with an input to the first layer by the taking transaction features of source node $X^{txn}_{\tau(v_s)}$, its node type embedding $\tau(v_s)^{emb}$ and the edge type embedding $\phi(e)^{emb}$ to calculate $V^i(v_s) = \text{V-Linear}^i_{\tau(v_s)}\left(X^{txn}_{\tau(v_s)} + \tau(v_s)^{emb} + \phi(e)^{emb}\right)$, Then, for $H^{(l-1)}$, where $l \neq 1$, we compute $V^i(v_s) = \text{V-Linear}^i_{\tau(v_s)}\left(H^{(l-1)}[v_s]\right)$.

We adopt the multiheaded attention to control the randomness of initial weights. First, we compute the attention output of one attention head, denoted by $\alpha\text{-head}^i(v_s, e, v_t)$, using this equation $\alpha\text{-head}^i(v_s, e, v_t) = \frac{\left(K^i(v_s)W^{att}_{\tau(v_s)} + Q^i(v_t)W^{att}_{\tau(v_t)}\right)}{\sqrt{d_k}}$, where $\sqrt{d_k}$ is the square root of the key vector's dimension.

The heterogeneous mutual attention of the target node query vector $Q^i(v_t)$ and the source node key vector $K^i(v_s)$ is then computed by $\alpha(v_s, e, v_t) = \underset{\forall v_s \in N(v_t)}{\text{softmax}}\left(\underset{i \in [1,h]}{\|} \alpha\text{-head}^i(v_s, e, v_t)\right)$, where $N(v_t)$ represents the neighbors of v_t, h the number of attention heads, $\|$ vector concatenation.

Finally, the message passing between $H^{(l)}$ and $H^{(l-1)}$ is given by $\text{msg}(v_s, e, v_t) = \underset{i \in [1,h]}{\|}\left(V^i(v_s) \cdot \text{dropout}\left(\alpha\text{-head}^i(v_s, e, v_t)\right)\right)$, where the right hand side is a concatenation of all msg-head^i, the message passing of one attention head at the ith query vector.

3.2.3 *xFraud Detector+: An Improvement over HGT.*

We implement xFraud detector and detector+, whose difference lies in the sampler. In detector, we use the original HGT implementation[2] and empirically show that HGSampling is computationally costly (see the inference time in Figure 10). Hence, we modify the sampling as in GraphSAGE and denote the efficient version of the xFraud detector as detector+. In detector+, the algorithm first samples

[2]https://github.com/acbull/pyHGT (last accessed: Oct 18, 2020).

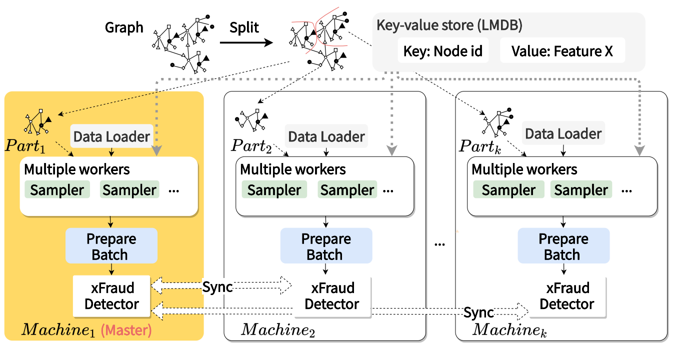


Figure 5: Architecture of Distributed xFraud Detector+.

k-hop neighborhood of a node and then aggregates feature information from neighbors and finally allows GNN to predict the label using aggregated information. In HGSampling, used by HGT, it tries to maintain a similar number of different $\tau(v)$ and $\phi(e)$ types after sampling and minimize the information loss and sample variance in the subgraph after sampling. However, in our datasets, the graph is much sparser (2.12 and 1.49 edges/node for *eBay-small* and *eBay-large*) compared with the Open Academic Graph (11.173 edges/node) used in HGT. Therefore, HGSampling is more costly than GraphSAGE because it requires all types of nodes and edges to be of similar size in the sampled subgraph. In the following sections, we consider xFraud detector equivalent to HGT and mainly focus on the evaluation of xFraud detector+.

3.3 Distributed xFraud Detector+

To make xFraud detector+ scalable to industrial-scale datasets, we designed a distributed learning architecture (see Figure 5). We briefly discuss its design and leave more details to Appendix C [34].

3.3.1 ***Graph Partitioning.*** We adopt the Power Iteration Clustering (PIC) algorithm [25] to partition the graph according to pairwise similarities of edge properties. PIC is effective for graph partition/clustering and well-suited to very large datasets due to its high efficiency. Each worker takes charge of a different partition during distributed training.

3.3.2 ***Distributed Learning.*** We utilize the DistributedDataParallel (DDP) tool provided by the flexible package, *PyTorch Ignite* [12], for distributed model learning. In terms of gradient synchronization, the gradients computed by different workers will be averaged following the default DDP gradient synchronization protocol. After that, parameters of the local model will be updated, and all models on different workers will be the same.

3.3.3 ***Data Loading.*** We use a lightweight KV-store to store all graph-related information. We choose to use Lightning Memory-Mapped Database (LMDB) as it allows us to have multiple data loaders simultaneously, where each worker has its own data loader. This alleviates the system bottleneck that we had when using LevelDB for the same purpose, which we found challenging to support multi-thread operations. This design decision turns out significant in reducing the training and inference time.

Table 1: Top$k$ hit rate computed on different explainability methods using various measures (on all 41 communities).

	Measures to calculate hit rate	H_{Top5}	H_{Top10}	H_{Top15}	H_{Top20}	H_{Top25}
1	edge betweenness	**0.469**	0.718	0.812	**0.903**	0.923
2	edge load	0.455	0.707	0.812	0.902	0.923
3	approximate current flow betweenness	0.450	0.690	**0.821**	0.899	0.923
4	betweenness	0.451	**0.724**	0.815	0.901	0.923
5	closeness	0.464	0.719	0.816	0.901	**0.924**
6	communicability betweenness	0.448	0.688	0.812	0.899	0.922
7	current flow betweenness	0.446	0.700	0.820	0.900	0.922
8	current flow closeness	0.441	0.691	0.815	0.900	**0.924**
9	degree	0.464	0.716	0.815	0.901	**0.924**
10	eigenvector	0.443	0.714	0.811	0.901	**0.924**
11	harmonic	0.464	0.719	0.816	0.901	**0.924**
12	load	0.452	**0.724**	0.815	0.901	0.923
13	subgraph	0.447	0.714	0.813	0.899	0.922
14	GNNExplainer weights	0.445	0.692	**0.821**	0.898	0.921
15	random weights	0.127	0.454	0.602	0.695	0.791

3.4 The Explainer of xFraud

We present a hybrid explainer (see Figure 4) in xFraud based on a trade-off between the *task-aware* GNNExplainer and the *task-agnostic* centrality measures.

3.4.1 ***Trade-off between GNNExplainer and edge centrality.*** GNNExplainer [47] is a model-agnostic explainer which assigns edge weights during node prediction. But, a fundamental question is — *Is GNNExplainer itself optimal for all scenarios? What if we replace the edge weights with other measures, such as random weights and edge centrality?* To answer this, we conduct a micro benchmark against other measures (see Table 1). For conducting a fair comparison, we design a metric called Top$k$ hit rate.

Metric: Top*k* hit rate. We create human annotations of edge importance and compare the explanation weights with the human annotations. The goal of this quantitative evaluation is to quantify the agreement between different edge importance measures and human annotations. Note that the average edge importance scores and edge weights are in different domains (see Figure 6 for an example): the former are discrete values $\in [0, 2]$, and the latter continuous numbers $\in [0, 1]$. We need a metric that reports the edge ranking of the most important ones in both domains. Hence, we compute the top$k$ hit rate H, defined as $\frac{\mathbb{N}^{+,k}_{human} \cap \mathbb{N}^{+,k}_{explainer}}{k}$, where $\mathbb{N}^{+,k}_{\_}$ denotes the set of edges ranked by human/explainer as top k. Concretely, we count the common edges in their top k selection and divide this count by k.

Trade-off and Intuition. Table 1 and Figure 7 illustrate a trade-off between GNNExplainer and edge centrality measures—they work well on different "communities" (test examples) and none of them dominates the other. GNNExplainer is developed to explain the predictions generated by a GNN network. GNNexplainer computes the importance scores of node features and assigns edge weights, with which we determine the most informative edges when a node prediction is made. On the contrary, edge centrality measures are popular methods to quantify the edge importance in a network, which is task-agnostic for node prediction. Intuitively, we should combine these two measures to generate an explainer which attends to both task-aware and task-agnostic measures.

3.4.2 ***xFraud Hybrid Explainer.*** Based on the above analysis, we propose a hybrid explainer and formulate a learning problem as follows. First, we learn two coefficients, namely, the centrality

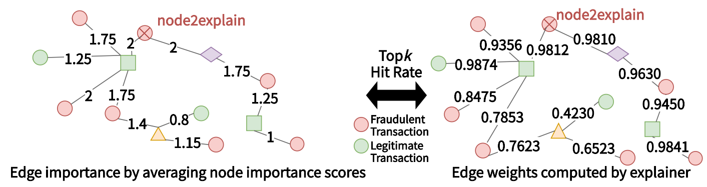

node2explain
Edge importance by averaging node importance scores
Topk Hit Rate
Fraudulent Transaction
Legitimate Transaction
node2explain
Edge weights computed by explainer

Figure 6: Edge importance scores (left) by human and edge weights by the explainer (right).

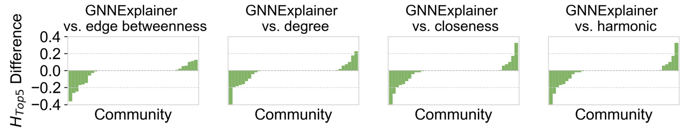


Figure 7: GNNExplainer and centrality measures work well for different communities, forming a trade-off.

coefficient A and explainer coefficient B to combine the weights from different explanation mechanisms: $A * w(c) + B * w(e)$. We can learn these two coefficients by either Ridge regression or directly maximizing the hit rate on the training set.

4 EXPERIMENTS OF XFRAUD DETECTOR+

We conduct extensive experiments on real-world transaction datasets sampled from the eBay commerce platform to verify the efficacy and efficiency of xFraud detector+. The statistics of the datasets are summarized in Table 2. The details on the graph construction process are in Appendix B [34]. We run end-to-end experiments on *eBay-xlarge* as it is a superset of *eBay-large* and *eBay-small*. Specifically, we run the distributed version of xFraud detector+ since *eBay-xlarge* is too large to be fit into a single machine. In the graph partition process, we first split the whole graph into 128 subgraphs using PIC. We then organize these 128 subgraphs into κ groups, where κ is the number of workers[3]. Different groups are handled by different workers. After the end-to-end evaluation (Sec. 4.1), we conducted an ablation study (Sec. 4.2) on *eBay-large* and *eBay-small* to study the trade-off between xFraud detector (i.e., HGT) and xFraud detector+.

4.1 End-to-end Experiments

We report the end-to-end results on *eBay-xlarge* in Table 3. In addition, we show precision-recall curve and ROC curve in Figure 8 and Figure 9 to further study prediction performance as *eBay-xlarge* is an extremely imbalanced graph dataset.

End-to-end Results. From Table 3, our detector+, achieves the best AUC (averaged across seeds) using 8 machines w.r.t. GEM and GAT. In terms of training efficiency, xFraud detector+ takes only slightly longer time per epoch compared to GEM in an 8-machine setting. If we increase the number of machines to 16, the training time is reduced by 1.89×, 1.84× and 1.82× for GAT, GEM, and our detector+, respectively. Despite roughly linear speedups, we observe

[3]We first order the 128 subgraphs according to the total number of nodes in ascending order. Then, we put the first few subgraphs that cumulatively have $\lceil \frac{|\mathcal{V}|}{\kappa} \rceil$ nodes into the same group. We repeat this process until we get κ groups. In this way, we ensure that each machine receives a graph partition of similar total number of nodes.

Table 2: Dataset summary ("B":billion;"M":million;"K":thousand)
***The ratio of frauds is only reported on the sampled datasets.**

Dataset	Features	Graph type	#Nodes	#Edges	Fraud%*
eBay-xlarge	480	hetero	1.1B	3.7B	4.33%
eBay-small	114	hetero	289K	613K	4.30%
eBay-large	480	hetero	8.9M	13.2M	3.57%

Table 3: End-to-end performance on the dataset *eBay-xlarge* (epochs: 128). We report the average scores over two different seeds (A and B).

# machines	Model	AUC	Training time (s/epoch)	Inference time (s/batch)
8	GAT	0.8879	62.74	0.0557 ± 0.1966
	GEM	0.8961	**61.77**	**0.0167** ± **0.0054**
	xFraud detector+	**0.9074**	70.47	0.0799 ± 0.1868
16	GAT	0.8866	**33.11** (1.89×)	0.0557 ± 0.1966
	GEM	**0.8938**	33.56 (1.84×)	**0.0167** ± **0.0054**
	xFraud detector+	0.8892	38.72 (1.82×)	0.0799 ± 0.1868

lowered AUC compared with 8 machines. In our implementation, each machine only has a subgraph; therefore, all three methods obtain a suboptimal model due to a restrained field of neighbors and edges. This phenomenon reveals a trade-off about our current way of handling large-scale graphs — one can use more resources to accelerate the model training but might have to compromise the model performance. It is an interesting future work to understand how to develop better distributed algorithms for training heterogeneous graph models. For more details on the system implementation and results, see Appendix C [34]. GEM (8 machines) takes the shortest time to do inference over a batch of 640 nodes due to the simplicity of its convolution layers. GAT and xFraud have longer inference time because their implementations of attention mechanisms. xFraud takes slightly longer than GAT due to its attention on heterogeneous types of nodes and edges. Since all methods take less than 0.1 second for a batch, all of them are practical to be deployed in production. Overall, xFraud is appealing in fraud detection, as it achieves the best model quality with a reasonably fast inference speed. Since xFraud detector+ is scalable to 16 machines, an online production scenario using xFraud can leverage historical and up-to-date transaction records to incrementally train a detector (Appendix H [34]).

Precision-recall Curve (P/R Curve). Figure 8 illustrates the P/R Curve using different settings. The trade-off of precision and recall is an ever-lasting goal for machine learning models; and xFraud detector+ achieves a better balance between precision and recall compared to GAT and GEM, which means that our model can return more accurate results (higher precision) as well as most of the true fraudulent transactions being found (higher recall).

ROC Curve. In *eBay-xlarge*, the majority of transactions are benign and the ratio of fraud transactions is very low. Besides, from the application perspective, the task of fraud detection is vulnerable to false positive cases, when benign transactions are flagged as fraud. This would cause an overwhelming human verification and significantly worsen user experience. Therefore, it is important to study the imbalance-aware metrics like true positive rate (FPR) and false positive rate (TPR). Specifically, if we restrict the FPR

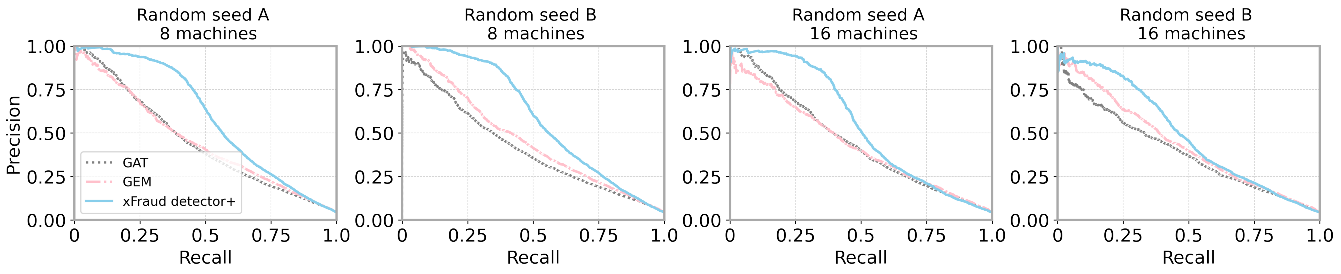


Figure 8: Precision-recall curves using different settings (seeds and # machines) on *eBay-xlarge*.

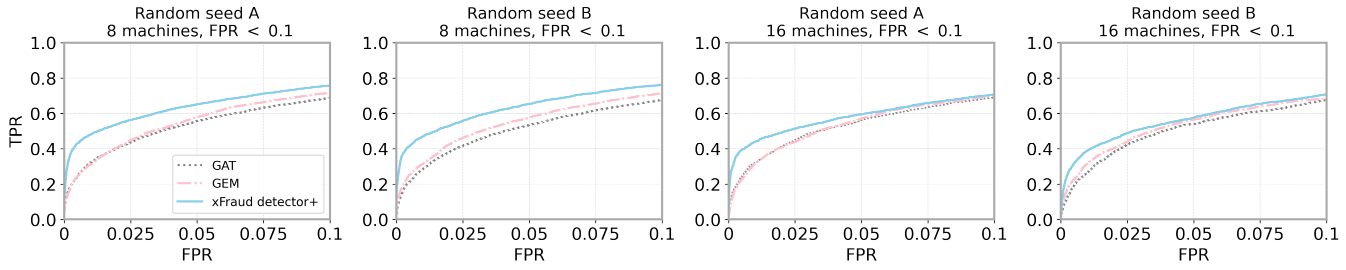


Figure 9: ROC curves using different settings (seeds and # machines) on *eBay-xlarge* for FPR < 0.1.

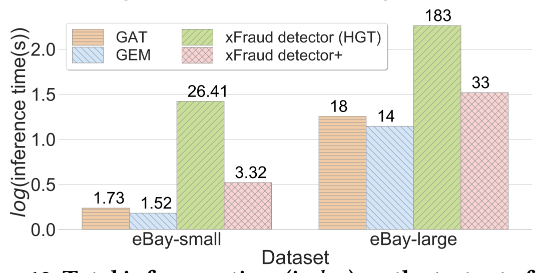


Figure 10: Total inference time (in *log*) on the test set of *eBay-small* and *eBay-large* (actual time in parentheses).

being lower than 0.1 as in Figure 9 (even this small ratio could involve 85.7M transactions in *eBay-xlarge*), xFraud significantly outperforms GAT and GEM when only a small FPR is allowed. We plot the full range of FPR in Figure 15 of Appendix H [34], where the three models have a similar area under ROC curve (i.e., AUC-ROC), xFraud's ROC curve is consistently beyond GAT and GEM.

Discussion. All results we report are on a dataset after pre-filtering the fraudulent/benign transactions with rule/ML-based filters and down-sampling the benign transactions. In Appendix H [34] we discuss the implication of this pre-filtering step and the production scenario of xFraud. As we will see, even without downsampling benign transactions, xFraud achieves a reasonable precision and recall for fraud detection on industrial-scale data. E.g., of the 3 fraud candidates investigated by the business unit, 1 will be a real fraud, with 0.1 of recall.

4.2 Ablation Study: xFraud detector+ vs. xFraud detector (i.e., HGT)

Here, we conduct an ablation study of xFraud detector (i.e., HGT) and xFraud detector+ to demonstrate the efficiency of the sampler. We run experiments on *eBay-small* and *eBay-large*, which are subsets of *eBay-xlarge* because our *eBay-xlarge* is too large such that xFraud detector (i.e., HGT) can no longer handle it. We report the inference time and AUC using a single machine in Figure 10. Concretely, xFraud detector+ achieves a 5 × speedups in terms of inference time (during testing) on *eBay-large* compared with xFraud detector (i.e., HGT), and the speedup on *eBay-small* is even larger, i.e., up to 7 ×. Meanwhile, using a simplified yet efficient sampler (GraphSAGE) will not sacrifice the model AUC (*eBay-small* vs. *eBay-large*): 0.7248 vs. 0.8683 for HGT, and 0.7262 vs. 0.8690 for xFraud detector+. Interestingly, we observe that the xFraud detector+ can even slightly outperform xFraud detector (i.e., HGT) on both datasets in terms of AUC.

5 EXPERIMENTS OF XFRAUD EXPLAINER

In this section, we discuss how we build an xFraud explainer on top of the detector. The main contribution of the explainer is to compute node features and edge masks of important features and nodes. As we see in the quantitative analysis (Sec. 5.1) and the case studies (Sec. 5.2), xFraud explainer ranks important edges with high agreement with expert human annotators and provides interesting insights for risk experts when analyzing risk propagation in a local heterogeneous graph. The output of the explainer carries the following meaning: node feature masks give high weights to the node feature dimensions influential in prediction; edge masks are the weights of edges in the subgraph, which indicate the strength of connectedness between pairs of nodes when flagging fraud. We visualize the subgraph structure with these outputs.

5.1 Quantitative Analysis of xFraud Explainer: Top*k* Hit Rate

First, we want to quantify the efficacy of xFraud explainer by studying the agreement between human perception and explainer output. Our approach is to compute the agreement between the edge importance scores based on human annotations and the edge weights generated by our hybrid explainer.

Table 4: Top$k$ hit rate in the test communities.

H(_)	Edge betweenness $H(c)$	GNNExplainer $H(e)$	Hybrid (ridge) $H(h)$	Hybrid (grid) $H(h)$
Top5	0.45540	0.44800	0.44890	**0.45550**
Top10	0.78175	0.77580	**0.81115**	0.78700
Top15	0.87763	0.88473	0.88963	**0.89410**
Top20	0.96205	0.95840	0.96198	**0.96275**
Top25	**0.96616**	0.95954	0.96614	0.96614

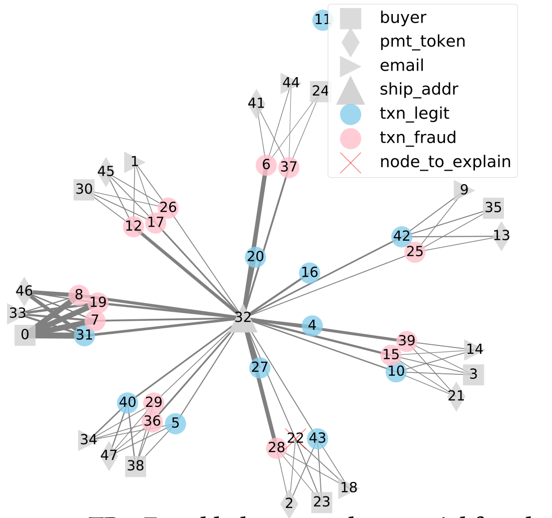


Figure 11: TP: xFraud helps to catch potential frauds.

Sample. In total, we randomly select 41 communities from our test set, among which 18 communities has a fraudulent transaction as seed (label 1), 23 a legitimate transaction as seed (label 0). The AUC score of this test sample is 81.88%. A community is formed around a transaction seed node, where all connected nodes and edges are taken. In total, we have 1,591 nodes of five types (buyer, transaction, shipping address, email, and payment token), and 3,344 edges.[4] On average, there are 81.56 edges per community.

Human annotations and edge importance score. We have created the human evaluation of edge importance scores of all edges in these 41 communities. The annotation protocol and score calculations are listed in Appendix E [34]. Out of 41 communities, we take the first 21 communities as the training set, the last 20 as the test set. We trained two versions of the hybrid explainer: (1) via ridge regression on the human annotations on the training set, and (2) via directly optimizing the average hit rate on the training set.

Results. Table 4 illustrates the result. We see that the hybrid explainer consistently outperforms both GNNExplainer and centrality measures. This is not surprising, as shown in our previous discussion of the tradeoff. It is an exciting future direction to come up with better ways to combine these different metrics together to form an even better explainer for graphs.

5.2 Qualitative Analysis of xFraud Explainer: Case Studies

To visualize the subgraph for a certain node, we use the node index, edge indices and their masks, true labels of nodes as input. The thicker an edge is, the stronger the connection is. We visualize the connections with nondirectional edges and use the ground truth labels for transactions. **True positive (TP): flagging frauds.** In Figure 11, we see a generic shipping address (node 32, a warehouse) connected to both fraudulent/benign transactions related to various buyers using various payment tokens/emails. According to BU, one explanation for this pattern is there is often a lag between user chargebacks and when the frauds have taken place, not to mention it is possible that a card stolen claim might never be forwarded to eBay from some banks[5]. As a result, we cannot fully trust the positive labels in such a case, where it is clearly unusual for such a community to have an extremely mixed benign/transactions across buyers. And it could also be a case where defaulters disguise their true purposes by "cultivating" some legitimate accounts to execute a few legit transactions. For the 2^{nd} assumption, the BU needs extra evidence to further examine. Either way, xFraud is able to flag the node-to-predict as fraudulent by learning from the important edges (the thicker ones), and to inform the BU that this set of buyers are highly suspicious and should be under more detailed examinations. This shows the importance in detecting frauds on the transaction level as we propose in xFraud, instead of just on the account level as in GEM [28]. Currently, the BU is only using a rule based system[6] to filter the suspicious transactions stored in the tabular format. xFraud explainer is innovative to a traditional BU annotation routine, because it allows experts to combine graph level and feature level information. For extensive case studies on **false positive (FP): benign → fraud** and **false negative (FN): fraud → benign**, we discuss in Appendix G [34], where we also discuss system limitations and potential solutions to improve xFraud.

6 CONCLUSION

In this paper, we propose xFraud, a system for detecting fraud transaction and explaining model prediction. Specifically, a heterogeneous graph is constructed and a self-attentive heterogeneous graph neural network is leveraged for risky transaction scoring. We further design a learnable hybrid explainer that leverages both GNNExplainer and centrality measures to learn node- and edge-level explanations simultaneously. Through extensive experiments on real-world datasets, we show the proposed xFraud detector+ can efficiently process billion-scale heterogeneous graphs and outperform the competitive baselines. More importantly, xFraud is the first work that quantifies a strong agreement between human perception and explainer outputs. Real-world case studies illustrate that with the hybrid xFraud explainer, we can generate convincing explanations to assist further decision-making of business units.

Acknowledgments. CZ and the DS3Lab gratefully acknowledge the support from Swiss National Science Foundation (Project Number 200021 _184628 and 197485), Innosuisse/SNF BRIDGE Discovery (Project Number 40B2-0_187132), European Union Horizon 2020 Research and Innovation Programme (DAPHNE, 957407), Botnar Research Centre for Child Health, Swiss Data Science Center, Alibaba, Cisco, eBay, Google Focused Research Awards, Kuaishou Inc., Oracle Labs, Zurich Insurance, and the Department of Computer Science at ETH Zurich.

[4]Note that explainer assumes directions and assigns two weights to bidirectional edges connecting a pair of nodes. Since human annotation is on the node level, and it is generally hard for annotators to consider directions, we remove directions in the explainer weights by taking the larger weight.

[5]This is out of control of eBay and is due to the inconsistency of reporting systems at some banks.

[6]*skope-rules* which perform rule mining on tabular data, https://github.com/scikit-learn-contrib/skope-rules (last accessed: Oct 18, 2020).

REFERENCES

[1] Bart Baesens, Veronique Van Vlasselaer, and Wouter Verbeke. 2015. *Fraud analytics using descriptive, predictive, and social network techniques: a guide to data science for fraud detection.* John Wiley & Sons.

[2] Federico Baldassarre and Hossein Azizpour. 2019. Explainability techniques for graph convolutional networks. *arXiv preprint arXiv:1905.13686* (2019).

[3] Adam Breuer, Roee Eilat, and Udi Weinsberg. 2020. Friend or Faux: Graph-Based Early Detection of Fake Accounts on Social Networks. In *Proceedings of The Web Conference 2020.* 1287–1297.

[4] Bokai Cao, Mia Mao, Siim Viidu, and S Yu Philip. 2017. HitFraud: a broad learning approach for collective fraud detection in heterogeneous information networks. In *2017 IEEE international conference on data mining (ICDM).* IEEE, 769–774.

[5] Shaosheng Cao, XinXing Yang, Cen Chen, Jun Zhou, Xiaolong Li, and Yuan Qi. 2019. TitAnt: online real-time transaction fraud detection in Ant Financial. *Proceedings of the VLDB Endowment* (2019).

[6] Yukuo Cen, Xu Zou, Jianwei Zhang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Representation learning for attributed multiplex heterogeneous network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* 1358–1368.

[7] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. 2015. Heterogeneous network embedding via deep architectures. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining.* 119–128.

[8] Sarthika Dhawan, Siva Charan Reddy Gangireddy, Shiv Kumar, and Tanmoy Chakraborty. 2019. Spotting collective behaviour of online frauds in customer reviews. *arXiv preprint arXiv:1905.13649* (2019).

[9] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining.* 135–144.

[10] Paul Emmerich, Maximilian Pudelko, Sebastian Gallenmüller, and Georg Carle. 2017. FlowScope: Efficient packet capture and storage in 100 Gbit/s networks. In *2017 IFIP Networking Conference (IFIP Networking) and Workshops.* IEEE, 1–9.

[11] Dhivya Eswaran, Stephan Günnemann, Christos Faloutsos, Disha Makhija, and Mohit Kumar. 2017. Zoobp: Belief propagation for heterogeneous networks. *Proceedings of the VLDB Endowment* 10, 5 (2017), 625–636.

[12] V. Fomin, J. Anmol, S. Desroziers, J. Kriss, and A. Tejani. 2020. High-level library to help with training neural networks in PyTorch. https://github.com/pytorch/ignite.

[13] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020.* 2331–2341.

[14] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems.* 1024–1034.

[15] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. Fraudar: Bounding graph fraud in the face of camouflage. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* 895–904.

[16] Binbin Hu, Yuan Fang, and Chuan Shi. 2019. Adversarial learning on heterogeneous information networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* 120–129.

[17] Binbin Hu, Zhiqiang Zhang, Chuan Shi, Jun Zhou, Xiaolong Li, and Yuan Qi. 2019. Cash-out user detection based on attributed heterogeneous information network with a hierarchical attention mechanism. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 946–953.

[18] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020.* 2704–2710.

[19] Qiang Huang, Makoto Yamada, Yuan Tian, Dinesh Singh, Dawei Yin, and Yi Chang. 2020. GraphLIME: Local interpretable model explanations for graph neural networks. *arXiv preprint arXiv:2001.06216* (2020).

[20] Parisa Kaghazgaran, James Caverlee, and Anna Squicciarini. 2018. Combating crowdsourced review manipulators: A neighborhood-based approach. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining.* 306–314.

[21] Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian. 2018. Rev2: Fraudulent user prediction in rating platforms. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining.* 333–341.

[22] Ao Li, Zhou Qin, Runshi Liu, Yiqun Yang, and Dong Li. 2019. Spam review detection with graph convolutional networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management.* 2703–2711.

[23] Xiang Li, Wen Zhang, Jiuzhou Xi, and Hao Zhu. 2018. HGsuspector: Scalable Collective Fraud Detection in Heterogeneous Graphs. (2018).

[24] Chen Liang, Ziqi Liu, Bin Liu, Jun Zhou, Xiaolong Li, Shuang Yang, and Yuan Qi. 2019. Uncovering Insurance Fraud Conspiracy with Network Learning. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval.* 1181–1184.

[25] Frank Lin and William W Cohen. 2010. Power iteration clustering. In *ICML.*

[26] Shenghua Liu, Bryan Hooi, and Christos Faloutsos. 2017. Holoscope: Topology-and-spike aware fraud detection. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management.* 1539–1548.

[27] Ziqi Liu, Chaochao Chen, Longfei Li, Jun Zhou, Xiaolong Li, Le Song, and Yuan Qi. 2019. Geniepath: Graph neural networks with adaptive receptive paths. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4424–4431.

[28] Ziqi Liu, Chaochao Chen, Xinxing Yang, Jun Zhou, Xiaolong Li, and Le Song. 2018. Heterogeneous graph neural networks for malicious account detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management.* 2077–2085.

[29] Zhiwei Liu, Yingtong Dou, Philip S Yu, Yutong Deng, and Hao Peng. 2020. Alleviating the Inconsistency Problem of Applying Graph Neural Network to Fraud Detection. *arXiv preprint arXiv:2005.00625* (2020).

[30] Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. 2021. Are we really making much progress? Revisiting, benchmarking, and refining heterogeneous graph neural networks. (2021).

[31] Jun Ma, Danqing Zhang, Yun Wang, Yan Zhang, and Alexey Pozdnoukhov. 2018. GraphRAD: A Graph-based Risky Account Detection System. (2018).

[32] Wei Min, Zhengyang Tang, Min Zhu, Yuxi Dai, Yan Wei, and Ruinan Zhang. 2018. Behavior language processing with graph based feature generation for fraud detection in online lending. In *Proceedings of Workshop on Misinformation and Misbehavior Mining on the Web.*

[33] Hamed Nilforoshan and Neil Shah. 2019. SliceNDice: Mining Suspicious Multi-Attribute Entity Groups with Multi-View Graphs. In *2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA).* IEEE, 351–363.

[34] Susie Xi Rao, Shuai Zhang, Zhichao Han, Zitao Zhang, Wei Min, Zhiyao Chen, Yinan Shan, Yang Zhao, and Ce Zhang. 2021. Appendix for xFraud: Explainable Fraud Transaction Detection. https://github.com/eBay/xFraud/blob/master/documents/Appendix_XFraud_VLDB.pdf.

[35] Yuxiang Ren, Hao Zhu, Jiawei ZHang, Peng Dai, and Liefeng Bo. 2019. EnsemFDet: An Ensemble Approach to Fraud Detection based on Bipartite Graph. *arXiv preprint arXiv:1912.11113* (2019).

[36] Chuan Shi, Binbin Hu, Wayne Xin Zhao, and S Yu Philip. 2018. Heterogeneous information network embedding for recommendation. *IEEE Transactions on Knowledge and Data Engineering* 31, 2 (2018), 357–370.

[37] Yu Shi, Fangqiu Han, Xinwei He, Xinran He, Carl Yang, Jie Luo, and Jiawei Han. 2018. mvn2vec: Preservation and collaboration in multi-view network embedding. *arXiv preprint arXiv:1801.06597* (2018).

[38] Kai Shu, Deepak Mahudeswaran, Suhang Wang, and Huan Liu. 2020. Hierarchical propagation networks for fake news detection: Investigation and exploitation. In *Proceedings of the International AAAI Conference on Web and Social Media*, Vol. 14. 626–637.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems.* 5998–6008.

[40] Haibo Wang, Chuan Zhou, Jia Wu, Weizhen Dang, Xingquan Zhu, and Jilong Wang. 2018. Deep structure learning for fraud detection. In *2018 IEEE International Conference on Data Mining (ICDM).* IEEE, 567–576.

[41] Jianyu Wang, Rui Wen, Chunming Wu, Yu Huang, and Jian Xion. 2019. Fdgars: Fraudster detection via graph convolutional networks in online app review system. In *Companion Proceedings of The 2019 World Wide Web Conference.* 310–316.

[42] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *The World Wide Web Conference.* 2022–2032.

[43] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. 2019. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591* (2019).

[44] Rui Wen, Jianyu Wang, Chunming Wu, and Jian Xiong. 2020. ASA: Adversary Situation Awareness via Heterogeneous Graph Convolutional Networks. In *Companion Proceedings of the Web Conference 2020.* 674–678.

[45] P. Reddy X. Li, J. Saude and M. Veloso. 2020. Classifying and understanding financial data using graph neural network. In *AAAI.*

[46] Carl Yang, Yuxin Xiao, Yu Zhang, Yizhou Sun, and Jiawei Han. 2020. Heterogeneous network representation learning: A unified framework with survey and benchmark. *IEEE Transactions on Knowledge and Data Engineering* (2020).

[47] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. Gnnexplainer: Generating explanations for graph neural networks. In *Advances in neural information processing systems.* 9244–9255.

[48] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. 2020. XGNN: Towards Model-Level Explanations of Graph Neural Networks. *arXiv preprint arXiv:2006.02587* (2020).

[49] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. *Advances in Neural Information Processing*

Systems 32 (2019), 11983–11993.
[50] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 793–803.
[51] Yiming Zhang, Yujie Fan, Yanfang Ye, Liang Zhao, and Chuan Shi. 2019. Key Player Identification in Underground Forums over Attributed Heterogeneous Information Network Embedding Framework. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 549–558.
[52] Kai Zhao, Ting Bai, Bin Wu, Bai Wang, Youjie Zhang, Yuanyu Yang, and Jian-Yun Nie. 2020. Deep adversarial completion for sparse heterogeneous information network embedding. In *Proceedings of the Web Conference 2020*. 508–518.
[53] Qiwei Zhong, Yang Liu, Xiang Ao, Binbin Hu, Jinghua Feng, Jiayu Tang, and Qing He. 2020. Financial Defaulter Detection on Online Credit Payment via Multi-view Attributed Heterogeneous Information Network. In *Proceedings of The Web Conference 2020*. 785–795.
[54] Yongchun Zhu, Dongbo Xi, Bowen Song, Fuzhen Zhuang, Shuai Chen, Xi Gu, and Qing He. 2020. Modeling Users' Behavior Sequences with Hierarchical Explainable Network for Cross-domain Fraud Detection. In *Proceedings of The Web Conference 2020*. 928–938.

A HETEROGENEOUS DATASETS IN THE LITERATURE

We list the heterogeneous datasets used in the literature in the past six years in the following table.

Table 5: A survey of the existing heterogeneous graph data sets ("B":billion;"M":million;"K":thousand).

Year	Paper	Dataset	# Node	# Edge	# Edge/# Node
2015	HNE (2015) [7]	BlogCatalog	5,196	171,743	33.05
2017	MVE [36]	PPI	16,545	1,098,711	66.41
		DBLP	69,110	1,884,236	27.26
		Youtube	14,901	13,552,130	909.48
		Twitter	304,692	131,151,083	430.44
		Flickr	35,314	6,548,830	185.45
2018	GEM [28]	GEM-graph	8M	10M	1.67
	HERec [36]	Yelp	95,110	488,120	5.13
		Douban Book	138,423	1,026,046	7.41
		Douban Movie	90,241	1,714,941	19.00
	metapath2vec [9]	DBIS	78,366	326,481	4.17
		AMiner CS	12,522,027	14,215,558	1.14
	mvn2vec [37]	Twitter	116,408	183,341	1.57
		Youtube	14,900	7,977,881	535.43
		Snapchat	7,406,859	131,729,903	17.78
2019	GATNE [6]	Alibaba-S	6,163	17,865	2.90
		Amazon-GATNE	312,320	7,500,100	24.01
		YouTube	15,088	13,628,895	903.29
		Twitter	456,626	15,367,315	33.65
		Alibaba	41,991,048	571,892,183	13.62
	GTN [49]	DBLP	26128	239,566	9.17
	HAN [42]	IMDB	21420	86,642	4.04
		ACM	10942	547,872	50.07
	HeGAN [16]	Yelp	3,913	38,680	9.88
		DBLP	37,791	170,794	4.52
		Aminer	312,776	599,951	1.92
		Movielens	10,038	1,014,164	101.03
	HetGNN [50]	Academic II	49,708	137,286	2.76
		Academic I	272,272	544,976	2.00
		CDs Review	123,736	555,050	4.49
		Movie Review	74,701	629,125	8.42
2020	HGT [18]	ogbn-mag	179M	2B	11.17
	HNE [46]	PubMed	63,109	244,986	3.88
	MAGNN [13]	LastFM-r	71,689	3,034,763	42.33
	MV-ACM [52]	Amazon	10,099	113,637	11.25
		Alibaba	40,324	149,587	3.71
		Twitter	40,000	1,028,364	25.71
		PPI	15,005	1,044,541	69.61
		Youtube	2,000	1,114,025	557.01
		Aminer	178,385	5,935,349	33.27
2021	HGB [30]	LastFM	20,612	141,521	6.87
		Amazon	10,099	148,659	14.72
		Freebase	180,098	148,659	0.83
		Movielens	43,567	539,300	12.38
		Amazon-book	95,594	846,434	8.85
		Yelp-2018	91,457	1,183,610	12.94
	xFraud (ours)	eBay-small	288,853	612,904	2.12
		eBay-large	8,857,866	13,158,984	1.49
		eBay-xlarge	1.1B	3.7B	3.36

B DATASET CONSTRUCTION

In the transaction records, there exist a rich set of relations when two transactions share some linkage entities, e.g., executed by the same buyer, using the same payment instrument, shipped to the same address.

Following the same graph construction protocol, we construct *heterogeneous graphs* out of the transaction records in different scales — *eBay-xlarge*, *eBay-large*, and *eBay-small*. Note that *eBay-large* and *eBay-small* are subsets of *eBay-xlarge*. The graph construction protocol is as follows. Both transaction and linkage entities are treated as nodes. If an entity is used in a transaction, we create an edge between the transaction node and that entity (see Sec. 3.1). Only transaction node has input features.

Table 6: Node type counts $|\mathcal{V}_t|$ in heterogeneous graphs.

Dataset	Node type	#Count	Node type%
eBay-xlarge	txn	857M	77%
	pmt	81M	7%
	email	72M	6%
	addr	62M	5%
	buyer	69M	5%
eBay-large	txn	3,752,225	42.40%
	pmt	1,180,114	13.30%
	email	1,307,179	14.80%
	addr	1,316,251	14.90%
	buyer	1,302,097	14.60%
eBay-small	txn	207,749	71.90%
	pmt	22,273	7.70%
	email	25,878	9.00%
	addr	7,138	2.40%
	buyer	25,815	9.00%

We construct three datasets from the historical transactions at eBay. Historical transaction records were sampled to generate three datasets with different scales. Note that the ratio of fraudulent transactions is usually much smaller than that of the legitimate ones; hence, we try to down sample the benign transactions to alleviate the class imbalance problem.

eBay-xlarge. It is a billion-scale dataset of 1.1 billion nodes and 3.7 billion edges, which contains seven months of selected transaction records. The dimension of transaction features is 480.

eBay-large. All the historical transaction records spanning a given period in *eBay-xlarge* are sampled. The dimension of transaction features is 480. The linking entities whose transaction numbers below a predefined threshold are removed to maintain graph connectivity, which means that there is no isolated transaction in the graph.

eBay-small. We take a subset of *eBay-large* to construct *eBay-small* by shrinking the transaction spanning period, and the dimension of transaction features is 114.

For all three datasets, to further reduce the graph size while preserving graph connectivity, we adopt a graph sampling strategy: (1) All fraudulent transactions and randomly sampled benign transactions are selected as seeds. (2) Each seed is expanded to its k-hop neighbors, and at each hop, no more than N neighbors are picked. (3) The neighborhoods around the seeds with transaction numbers less than five are filtered out. We have a similar graph sampling across all datasets: for *eBay-xlarge*, $k = 8, N = 512$, for *eBay-large* and *eBay-small* $k = 3, N = 32$.

In addition, for the *eBay-xlarge* dataset, we produce the training/testing labels via two filtering and sampling steps:

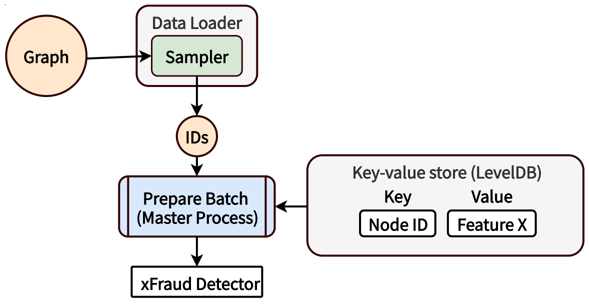


Figure 12: (Previous implementation) Using a single threaded KVStore to interact with GNN on a single machine.

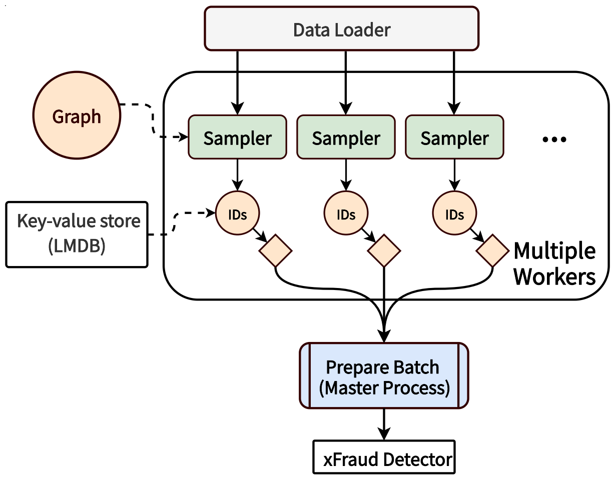


Figure 13: (New implementation) Using a multi threaded KV-Store to interact with GNN on a single machine.

(1) The original data stream has a fraud rate = 0.016%.
(2) We then use some simple rules to filter out certain low-risk transactions. These rules are already implemented in the eBay transaction platforms to filter transactions. This is similar to what GEM [28] does in graph preprocessing and is also consistent with how this model will be used in practice. After this step, we have a fraud rate = 0.043%.
(3) To create the training and testing labels, we sampled all fraud transactions, and 1% of non-fraud transactions. Note that the other transactions are still in the graph, but without supervised labels. This gives us the *eBay-xlarge* dataset with a fraud rate = 4.33%.

The ratio of transaction frauds is 4.33%, 3.57%, and 4.30%, in *eBay-xlarge*, *eBay-large*, and *eBay-small*. respectively. We further summarize the distribution of each node type in Table 6.

C MORE DETAILS ON THE DISTRIBUTED XFRAUD DETECTOR

We introduce the experimental protocols and an extensive set of results on our billion-scale dataset *eBay-xlarge*.

Implementation of KVStores. We have implemented two types of KVStores when optimizing xFraud detector+. The multi threaded KVStore solution is used in the distributed setting. We show the detailed implementation of the interaction between GNN and the single threaded KVstore and the multi threaded KVStore in Figure 12 and 13, respectively. With this new optimization, we manage to reduce the training (incl. data loading) on the *eBay-large* to 1 minute per epoch. In comparison, a single-threaded KVStore on the same dataset would take 45 min/epoch.

Further experimental details for detector.

- *Machines.* We conduct the single-machine experiments on a Linux server with 16 Intel Xeon Gold 6230 CPU, one Nvidia Tesla V100 32 GB GPU, and 256 GB memory. For the distributed settings, we use a set of machines, each of which has 4 Intel Xeon Gold 6230 CPU, one Nvidia Tesla V100 32 GB GPU, and 32 GB memory.
- *Hyperparameters.* The baselines and xFraud detector+ are trained with this set of hyperparameters: *n_hid* = 400, *n_heads* = 8, *n_layers* = 6, *dropout* = 0.2, *optimizer* = *"adamw"*, *clip* = 0.25, *max_epochs* = 128, *patience* = 32. We keep the same set of hyperparameters for all datasets.

Further experimental results. We present the complete results in Table 7 on two different seeds (A and B) using 8 machines and 16 machines.

Convergence on eBay-xlarge. In this paragraph, we discuss the model convergence under different settings. In Figure 14, the models trained on 16 machines do not converge faster than that on 8 machines. Also, the final AUCs on 16 machines are worse compared to those on 8 machines. In this case, training using 8 machines is a better choice across different models using the current graph partitioning strategy as described in Sec. 4.

D EXPERIMENTAL DETAILS OF A MODIFIED GNNEXPLAINER

We implement GNNExplainer using the trained xFraud detector, transaction features, node index (of node-to-explain), edge indices, edge types, and node types as input. We obtain node feature masks of the subgraph as output, as well as the edge masks. The hyperparameters of GNNExplainer are: *epochs* = 100, *lr* = 0.01, $\beta_{\text{edge\_size}}$ = 0.005, $\beta_{\text{node\_feature\_size}}$ = 1, $\beta_{\text{edge\_entropy}}$ = 1, $\beta_{\text{node\_feature\_size}}$ = 0.1. The xFraud detector is not retrained during the explanation process.

We extend the vanilla GNNExplainer [47] so that it generates node feature masks for all the nodes, which enables a node-level feature explanation to a prediction. Learning the feature importance of all node features was not possible with the vanilla GNNExplainer. The training of explainer is initialized with a random edge mask $1 \times |\mathcal{E}|$ and a random node feature mask $|\mathcal{V}| \times F$ (F the size of node features), as well as four random coefficients of edge size β_{es}, node feature size β_{nfs}, edge entropy β_{ee}, and node feature entropy β_{nfe}. It then takes the node index of node-to-explain, transaction features, node types, edge types and edge indices as input. The explainer takes the weights trained in the detector and uses only the detector model in the evaluation mode, as Figure 4 demonstrates (right). The loss function of explainer has to include edge entropy and node feature entropy so that the explainer knows which important nodes,

Table 7: Model performance in distributed settings on *eBay-xlarge* (# epochs: 128). The best score of a column is in bold.

Model	# of machines	Seed	Accuracy	AP	AUC	Training time (s/epoch)	Inference time (s/batch)
GAT	8	A	0.9334	0.4478	0.8894	62.62	0.0557 ± 0.1966
		B	0.9309	0.4120	0.8863	62.85	
	16	A	0.9348	0.4481	0.8894	**32.08**	
		B	0.9425	0.3932	0.8838	34.14	
GEM	8	A	0.9554	0.4513	0.8960	61.67	**0.0167± 0.0054**
		B	0.9578	0.4613	0.8962	61.86	
	16	A	0.9552	0.4349	0.8949	33.40	
		B	0.9574	0.4363	0.8926	33.71	
xFraud detector+	8	A	0.9701	**0.5942**	**0.9074**	71.79	0.0799 ± 0.1868
		B	**0.9703**	0.5931	0.9073	69.15	
	16	A	0.9694	0.5483	0.8900	37.35	
		B	0.9650	0.4932	0.8883	40.09	

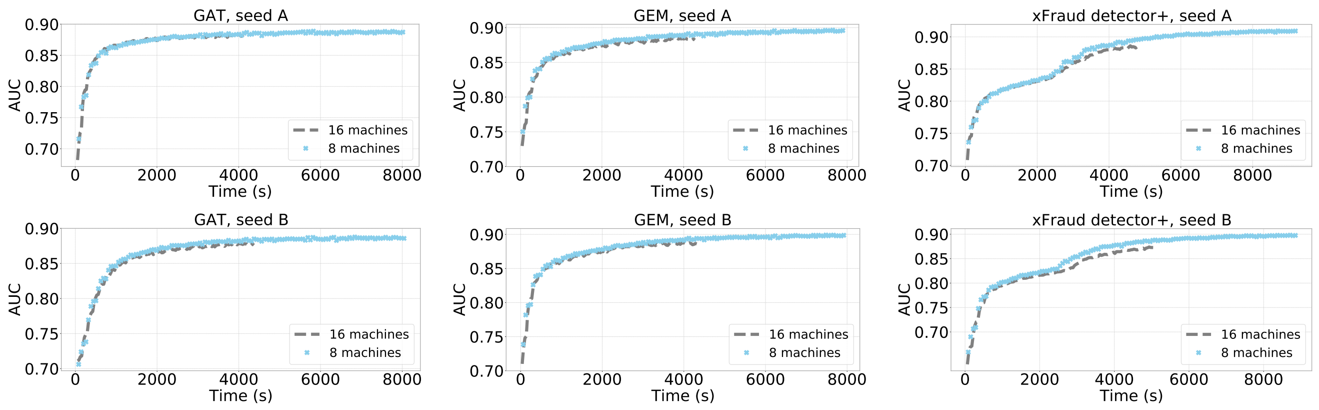


Figure 14: Convergence of distributed training (8 vs. 16 machines) on GAT, GEM, and xFraud detector+, measured by AUC (on two seeds).

node features and edges to attend in predicting the current node. The goal of explainer loss is to jointly minimize the detector loss with the entropy of node features and edges.

To compute the explainer loss, we first compute the edge mask and node feature mask of the subgraph S in G that contributes to the node-to-explain. We use M_{E_S} to denote the edge mask in the subgraph S (S a subgraph of G), and we have $M_{E_S} = \sigma(E_S)$, E_S a random initialization of edge parameters. Likewise, we denote the node feature mask as $M_{V_S} = \sigma(V_S)$, V_S a random initialization of node feature parameters. Now we calculate the explainer loss in three parts, detector loss (eq. 1), edge entropy (eq. 2), and node feature entropy (eq. 3). We finally sum them up to form the explainer loss:

$$\sum_i C_i \log(S_i), \tag{1}$$

$$\beta_{\mathrm{es}} \sum M_{E_S} + \beta_{\mathrm{ee}} \frac{\sum \Big(-M_{E_S} \log(M_{E_S} + \epsilon) - (1 - M_{E_S}) \log(1 - M_{E_S} + \epsilon) \Big)}{|\mathcal{V}_{\mathcal{E}}|}, \tag{2}$$

$$\beta_{\mathrm{nfs}} \frac{\sum M_{V_S}}{|\mathcal{V}_S|} + \beta_{\mathrm{nfe}} \frac{\sum \Big(-M_{V_S} \log(M_{V_S} + \epsilon) - (1 - M_{V_S}) \log(1 - M_{V_S} + \epsilon) \Big)}{|\mathcal{V}_S|}, \tag{3}$$

where $i \in$ {labeled transactions}, C_i is the true label of a transaction, S_i the output of the *softmax* layer in DNN, $|\mathcal{V}_S|$ and $|\mathcal{E}_S|$ the number of nodes and edges in the subgraph S, and ϵ a small constant to prevent $\log(0)$. By back propagation, the new loss is used to compute the new network weights in explainer and thus the network has the capacity to learn the subgraph nodes and their features that most importantly contribute to certain predictions made by the detector.

E ANNOTATION, NODE IMPORTANCE, RANDOM VS. GNNEXPLAINER

Annotation protocol. Now we introduce our annotation protocol of human ground truth. The goal of annotation is to assign a node importance score to each node in terms of how important the node is when the seed node prediction is made. We do not perform

Table 8: Top$k$ hit rate between explainer edge weights and human annotations: GNNExplainer vs. random.

Top$k$ hit rate	Top5	Top10	Top15	Top20	Top25
Random	0.13	0.45	0.60	0.70	0.79
GNNExplainer	0.45	0.69	0.82	0.90	0.92
$\Delta(GNNExplainer - Random)$	0.32	0.24	0.22	0.20	0.13

annotations on the edge level because the annotators only have access to node-level information (e.g., node features and representations). We have five expert annotators to ensure that each node is at least annotated by two annotators. There are three scales of node importance the annotators can assign: 0, 1, 2 for low, medium, and high importance, respectively. Then, we calculate the average inter-annotator agreement (IAA) score of annotator pairs. The average IAA score is 0.532 among 10 pairs of annotators, with the highest IAA 0.773 and lowest 0.314. But how do we know if our annotations are of high quality? Assuming the human annotators were randomly assigning scores, we employ random integer generators that assign 0, 1, 2 to replace all the annotations by human, and we calculate the average IAA score among the random annotators. Also, we repeat this random experiment 10 times and calculate the mean IAA score. The random IAA score is -0.006 which is significantly worse compared with 0.525, the IAA score of human annotators.

Node importance score. After we have obtained the human annotations by five annotators, we first calculate the average node importance score for a node v by $\frac{\sum annotation_i}{5}$, where an annotator $i \in [1, 2, 3, 4, 5]$. We then use these node importance scores to compute the edge importance scores. There are three strategies to calculate the edge importance based on its incident nodes — by averaging (“avg") or summing (“sum") the node importance scores, or by taking the minimal node importance score (“min"). There is no theoretical or empirical evidence of which aggregation method is the best, we therefore have run all the aggregation methods and select the one that performs the best.

Topk hit rate. In Figure 6, we demonstrate two graphs with edge importance scores and edge weights, respectively. We have to choose a meaningful set of k. Note that the average count of total edges across the communities is 81.56, among which the average count of edges with the largest edge importance is 20.90. This means that as long as k is smaller than or around 20, the top$k$ hit rate reflects a meaningful agreement score between the explainer and the annotators in ranking the important edges.

Random vs. GNNExplainer. Another caveat of the top$k$ edge selection is that we need to break the tie among the largest edge importance scores. To tackle this, we randomly draw 100 samples when selecting the top$k$ edges based on the highest importance score. Then, we take the average hit rate of these 100 draws. We have also run experiments with 10,000 draws, which renders similar results reported in Table 8. Here, we report the hit rates computed as the mean of random 100 draws.

In Table 8 we report the top$k$ hit rate with $k \in [5, 10, 15, 20, 25]$. Our hit rate is already 45% when inspecting the top 5 edges and increases as k increases. To draw the conclusion that the explainer agrees strongly with our human annotators, we look at a random baseline. The random baseline assumes that the explainer randomly assigns edge weights during prediction. We let a random number generator assign edge weights to each edge and we compute the top$k$ hit rate between random edge weights and edge importance scores. We repeat the random experiment 10 times, and finally obtain the average top$k$ hit rate from these 10 random draws. The random baseline is also reported in Table 8. Taking the difference between our hit rate and the random one, we see the largest discrepancy is at the top 5 hit rate and gradually drops as k increases. This demonstrates the strong agreement between the actual explainer weights and our human annotators.[7]

Comparing aggregation methods in analyzing explainer weights. There are three different aggregation strategies, ranging from node importance (human) to edge importance (human). In Tables 9, 10 and 11, there is no substantial difference in results generated by averaging, summing, or minimizing the weights. Also, in communities with a benign (0)/fraudulent (1) node-to-predict, there is no substantial difference. We also ran a paired-t-test to see if the distributions between these two types of communities are identical: the test results do not allow us to reject this null hypothesis on a significance level of 0.05. Therefore, in the follow-up experiments of hybrid explainer, we always use “averaging" as the aggregation method to compute the edge importance score (human).

Table 9: Top$k$ hit rate between explainer edge weights (“avg") and human annotations: GNNExplainer vs. random. $c1$: a community labeled 1, $c0$: a community labeled 0.

Top$k$ hit rate	Top5	Top10	Top15	Top20	Top25
Random	0.13	0.45	0.60	0.70	0.79
GNNExplainer	0.45	0.69	0.82	0.90	0.92
$\Delta(GNNExplainer - Random)$	0.32	0.24	0.22	0.20	0.13
$Random_{c0}$	0.31	0.20	0.21	0.20	0.15
$GNNExplainer_{c0}$	0.47	0.77	0.90	0.97	1.00
Δ_{c0}(GNNExplainer-Random)	0.16	0.57	0.69	0.27	0.85
$Random_{c1}$	0.33	0.17	0.23	0.21	0.11
$GNNExplainer_{c1}$	0.41	0.59	0.72	0.81	0.82
Δ_{c1}(GNNExplainer-Random)	0.08	0.42	0.49	0.60	0.71

F MORE DETAILS ON THE HYBRID EXPLAINER

Centrality measures as edge weights. We use the following two ways to compute edge weights using centrality measures:

(1) Compute the edge weights based on the node annotations using edge centrality measures such as **edge betweenness** and **edge load centrality**. Edge betweenness measures the number of the shortest paths between vertices that contain the edge, normalized by $\frac{2}{n(n-1)}$ in an undirected graph. Edge load counts the number of

[7]Comparing results from three aggregation methods (“sum" vs. “min" vs. “avg"), there is no significant difference in the distribution of our hit rates and $\Delta(GNNExplainer - Random)$. We hence report only the results of “avg". In addition, there is no substantial difference between $\Delta(GNNExplainer - Random)$ of communities labeled 1 and 0. For more details of the results on “min" and “sum" and their performances on communities labeled 1 and 0.

Table 10: Top$k$ hit rate between explainer edge weights ("min") and human annotations: GNNExplainer vs. random. $c1$: a community labeled 1, $c0$: a community labeled 0.

Top$k$ hit rate	Top5	Top10	Top15	Top20	Top25
Random	0.13	0.45	0.60	0.70	0.79
GNNExplainer	0.45	0.69	0.82	0.90	0.92
$\Delta(GNNExplainer - Random)$	0.32	0.24	0.22	0.20	0.13
$Random_{c0}$	0.16	0.48	0.69	0.77	0.85
$GNNExplainer_{c0}$	0.47	0.77	0.90	0.97	1.00
Δ_{c0}(GNNExplainer-Random)	0.31	0.29	0.21	0.20	0.15
$Random_{c1}$	0.08	0.42	0.49	0.60	0.71
$GNNExplainer_{c1}$	0.41	0.59	0.72	0.81	0.82
Δ_{c1}(GNNExplainer-Random)	0.33	0.17	0.23	0.21	0.11

Table 11: Top$k$ hit rate between explainer edge weights ("sum") and human annotations: GNNExplainer vs. random. $c1$: a community labeled 1, $c0$: a community labeled 0.

Top$k$ hit rate	Top5	Top10	Top15	Top20	Top25
Random	0.10	0.38	0.54	0.63	0.77
GNNExplainer	0.38	0.63	0.80	0.88	0.90
$\Delta(GNNExplainer - Random)$	0.28	0.25	0.26	0.25	0.13
$Random_{c0}$	0.17	0.40	0.61	0.70	0.85
$GNNExplainer_{c0}$	0.41	0.70	0.90	0.97	1.00
Δ_{c0}(GNNExplainer-Random)	0.24	0.30	0.29	0.27	0.15
$Random_{c1}$	0.03	0.37	0.45	0.56	0.67
$GNNExplainer_{c1}$	0.36	0.55	0.68	0.77	0.78
Δ_{c1}(GNNExplainer-Random)	0.33	0.18	0.23	0.21	0.11

the shortest paths which cross each edge. Then, we calculate the top$k$ hit rate between the edge centrality scores and the human annotations.
(2) Compute the edge weights based on the node annotations after transforming the original graph into a line graph. We compute various types of **node centralities** (e.g., closeness, eigenvector centrality, degree, etc.) measures in the line graph.

We report the results of representing edge weights with 13 centrality measures in Table 1. We compute the centrality measures using the *networkx*[8] package in Python.

Observations on comparing explainer weights and centrality measures. We observe that the hit rates computed by explainer weights $H(e)$ and by different centrality measures $H(c)$ are close. The similar hit rates show that GNNExplainer and centrality measures both manage to learn the most important edges when filtering the fraudulent transactions. For each rank, we mark the highest hit rate with bold: there is not a centrality measure that consistently outperforms the remaining measures. Even among various centrality measures, the $\Delta(H(e) - H(c))$ on different ranks vary. As k increases, the differences between the hit rates decrease.

We pick the best four centrality measures according to the hit rate, i.e., edge betweenness, degree, edge load, closeness and harmonic. If we examine the differences of hit rate $\Delta(H(e) - H(c))$ across communities in Figure 7, we notice that there is no clear winner between $H(e)$ and $H(c)$ in detecting the most important edges when comparing to human annotations. More importantly, a trade-off between $H(e)$ and $H(c)$ can be observed. This motivates us to learn a hybrid explainer using both explainer weights and centrality measures.

[8]https://networkx.org/documentation/stable/reference/algorithms/centrality.html (last accessed: Sep 1, 2021).

A hybrid learner that incorporates both explainer weights and centrality measures. We formulate the learning problem as follows. First, we learn two coefficients — centrality coefficient A and explainer coefficient B, which maximizes $avg(H(h))$ in the training communities via $avg(A * w(c) + B * w(e))$, where $w(c)$ denotes the centrality measures, and $w(e)$ the explainer weights. Since there are 41 communities, we take the first 21 communities as the training set and the last 20 as the test set. Then, we calculate $avg(H(h))$ in the test communities using the A and B learned over the training set. There are various techniques to optimize $avg(A * w(c) + B * w(e))$ and search for the optimal A and B, which maximize the average $H(h)$ in the training set. By taking a centrality measure that generates the best $H(c)_{Top5}$ in Table 1 (edge betweenness), we have run these three sets of experiments to optimize A and B:

(1) Fit polynomial functions where we find the best feature degree to maximize the average $H(h)$ in the training communities;
(2) Grid searches for A, where $B = 1 - A$;
(3) Train Ridge linear regressions on $avg(A * w(c) + B * w(e))$, where we also optimize the regularization hyperparameter α.

We run (1) to find the best degree d among {1, 2, ..., 9} and obtain $d = 1$ being the best fit, i.e., a linear combination of $A*w(c)+B*w(e)$. We illustrate the results of hybrid explainers constructed by (2) and (3) in Table 12. For the grid search in (2), we tune A in {0.00, 0.01, ..., 1.00}, where A maximizes the average hit rate in the training communities. For (3), the best α is 0.99 among {0.01, 0.02, ..., 0.99}, with $A = -0.1097, B = 0.1064$. If we compare the results of $H(h)$, $H(c)$, and $H(e)$ at all ranks, the hybrid learner has achieved at least a result as good as $H(c)$ if not better.

The results in Table 1 and Figure 7 validate that our proposed hybrid explainer can achieve a trade-off between the centrality measures and explainer weights. Both measures agree with human annotators, and we can leverage weights learned by explainer and centrality to construct a better explainer. This hybrid explainer incorporates both the topological features of the community and the message passing learned via GNNExplainer.

Moreover, we have also noticed that the centrality measures assign identical weights to many edges in the community. In contrast, the vanilla GNNExplainer always assigns a total order of weights to the edges, with the edges closer to the node-to-explain getting the highest weights. GNNExplainer offers a local explanation of the prediction, while centrality measures attend to the global structure of the community. Intuitively, by combining the best of both worlds, the hybrid explainer is beneficial since it considers both local and global structures of communities.

More Results on the Hybrid Explainer. We report the hit rate of top 5 edges across GNNExplainer, edge betweenness centrality, and the hybrid explainer trained using Ridge and grid search in Table 12.

Table 12: Top$k$ hit rate in the train and test communities by the hybrid explainer. A is the coefficient of centrality weights (edge betweenness) in the hybrid explainer $A * w(c) + B * w(e)$, where $B = 1 - A$.

H(_)	Edge betweenness $H(c)$		GNNExplainer $H(e)$		Hybrid (ridge) $H(h)$		Hybrid (grid) $H(h)$		
	Train	Test	Train	Test	Train	Test	Train	Test	A_{Train}
Top5	0.4817	0.45540	0.44257	0.44800	0.44648	0.44890	0.45971	**0.45550**	0.75
Top10	0.6581	0.78175	0.61210	0.77580	0.60767	**0.81115**	0.61881	0.78700	0.94
Top15	0.7497	0.87763	0.75981	0.88473	0.74838	0.89210	0.76375	**0.89410**	0.91
Top20	0.8459	0.96205	0.84126	0.95840	0.83505	0.96198	0.85595	**0.96275**	1.00
Top25	0.8813	**0.96616**	0.88350	0.95954	0.88286	0.96614	0.88379	0.96614	0.64
Top30	0.8868	0.96705	0.88778	0.95893	0.88790	**0.96780**	0.88938	**0.96780**	0.65
Top35	0.8920	**0.95780**	0.89388	0.94731	0.89339	0.95761	0.89444	0.95771	0.51
Top40	0.8976	0.93659	0.89860	0.92608	0.89893	0.93673	0.89917	**0.93764**	0.68
Top45	0.9026	0.91598	0.90244	0.90678	0.90443	**0.91608**	0.90472	0.91607	0.68

G CASE STUDIES USING HYBRID EXPLAINER

In this paper, a fraudulent transaction is marked as 1 (positive) and a benign one as 0. When we report true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), these four cases correspond to the following detection scenarios: "TP" – fraudulent transactions being flagged correctly, "TN" – benign transactions being flagged correctly, "FN" – we have missed frauds in detection, "FP" – benign transactions wrongly flagged as fraud.

G.1 More true positive cases

Except the case we show in Figure 16, we present six more TP cases in our 41 communities, visualized using weights learned in the hybrid explainer. We see from the visualization in (a), (c), and (f), xFraud does a good job in detecting suspicious transaction, even if the frequencies of fraudulent/benign cases are close. xFraud hybrid explainer not only manages to learn from the incident edges of the node-to-predicts, but also learns the path where the risk propagates. For instance, in (b), it learns that transaction 0 connects linking entities 17 (payment token) and 18 (email) with node 37 (buyer), which influences strongly the prediction of node 36 to be fraud. This type of risk propagation is useful in learning the embedding representations of linking entities. Similar to (b), we also observe a clear risk propagation path in (e). In case (d), we observe a typical fraudulent cluster which could be due to the fact that a valid payment token (node 19) was used at the beginning to gain the trust of the platform, then the defaulter conducts many fraudulent transactions. It could also be that the defaulter has hacked one payment token (node 5) and uses that payment token to conduct many transactions in a short time. In either case, the xFraud can assist BU to identify this kind of fraud cluster.

G.2 More case studies: false positives and false negatives

We investigate several case studies for the conditions "FP" and "FN" and share our insights.

False positive (FP): benign → fraud. In the community illustrated in Figure 17 (a), we have one buyer/email address/payment token and two shipping addresses. One shipping address (node 6) is much more frequently used than the other one. Note that this heavily used shipping address (node 6) is linked to many fraudulent transactions. This makes the prediction of seed node 39 prone to fraudulent (probability = 0.972 in xFraud detector+), because this transaction is directly linked to seed 6, which is considered highly suspicious by learning from its neighbors in the community. The most important edges in generating the prediction are marked in bold (0-6, 0-8). However, BU reports that this account is not fraudulent after investigation. Similarly, we have more fraudulent cases in the communities than the benign ones in the cases shown in Figure 17 (b) and 17 (c).

False negative (FN): fraud → benign. In a community like Figures 17 (d), where multiple buyers/payment tokens/email/shipping addresses are involved, we do not observe that one shipping address is used more frequently in the transaction logs. However, there are much more benign cases in the neighborhood, therefore xFraud detector+ has problems identifying a fraudulent transaction which could be due to a sudden attack, or a delay of user chargebacks. In Figure 17 (f) we see a simple (single-buyer) community with more fraudulent transactions than benign ones in the neighborhood. However, since the important (thick) edges are connected to more benign transactions than frauds (benign: 2, 14, 8, 9, 13, 11; fraud: 21, 17, 0), the node 1 is classified wrongly as benign.

To empirically investigate the case studies, we show in Table 13 that the different conditions are indeed correlated with communities types. We do not have false positives (benign wrongly recognized as fraudulent) in the complex communities, and xFraud does a better job in communities with more than one buyer.

To summarize the **FP: benign → fraud** cases, in all the FP cases we have (6 out of 6), there is only one buyer in the community, one linking entity of each type (i.e., one email, one payment token, one shipping address). Even if there are two linking entities, one is always more frequently used than the other one, and the seed is linked to the entity that is more frequently used. Whenever there is more fraudulent transactions connected to that linking entity, the benign node-to-predict is classified as fraudulent; vice versa for false negatives (2 out of 8). This is preemptive in risk prevention; the user might expect some short-term interruption of the service while being alert and notified by the platform that a fraudulent transaction might be ongoing.

In summary, the **FN: fraud → benign** cases show more variability, and our goal is to avoid as much as possible the false negatives. In many of the cases (6 out of 8), we see more buyers and linking entities (such as payment tokens and emails) in the community – we have higher false negatives in the complex communities. It is hard for xFraud detector+ to distinguish signals sent via various buyers and transactions. It could be deliberate ring attacks with accounts "cultivated" over a long time, or could be a one-time attack of one account using an unknown device.

G.3 System limitation and possible causes.

As we see in most of the cases, the misclassification of frauds is related to the ratio of fraudulent/benign cases in the k-hop neighborhood of the node-to-predict. It is therefore important to enforce a graph partition constraint of benign/fraudulent-ratio, so that the prediction is not strongly influenced by the frequency of cases. This goes into the direction we discussed before in the scalability experiments of *eBay-xlarge*: how to scale out training in a distributed setting on a heterogeneous graph.

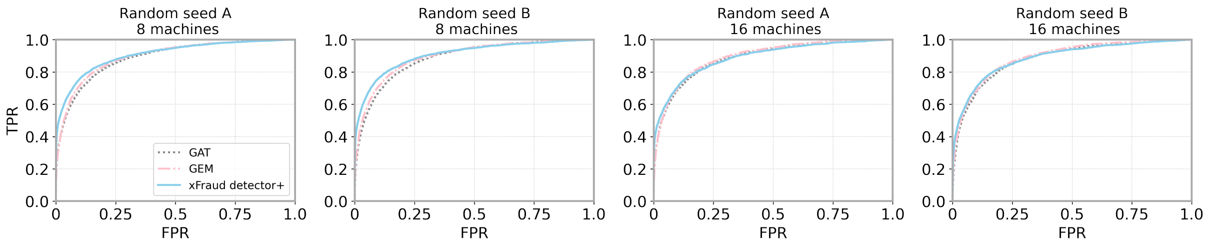


Figure 15: ROC curves using different settings (seeds and # machines) on *eBay-xlarge*.

Table 13: Confusion matrix of TP, TN, FP, and FN in simple and complex communities. The sample is the 41 communities we present in Sec. 5.1. We show the frequency and percentage of one condition in one type of communities in the bracket, where a simple community has only one buyer, and a complex one has more than one buyer. "TP": fraudulent transactions, "TN": benign transactions, "FN": missed frauds in detection, "FP": benign flagged as fraud.

Simple communities (16, 100%)		**Complex communities (25, 100%)**	
FP (6, 37.5%)	**TP** (4, 25%)	**FP** (0, 0%)	**TP** (6, 24%)
FN (2, 12.5%)	**TN** (4, 25%)	**FN** (6, 24%)	**TN** (13, 52%)

Although our system can flag some guest checkouts that are linked to suspicious entities, it still remains a difficult use case to capture with both ML and GNN methods. Guest checkout allows users to make purchases without logging and to stay anonymous. Our xFraud detector is designed for graph, therefore it requires purchases linked to some existing and nontrivial entities. Image a case where the user chooses a guest checkout to make a purchase. The email is newly registered, so it is not linked to any existing entities. The ip address is from a public places such as cafe or gas station. The payment token is a credit card which have not been used in our checkout system. The shipping address is a public transshipment warehouse. In such a case, none of the trivial entities can be linked by this purchase, so that our xFraud detector can hardly retrieve any useful information to make accurate predictions on the purchase.

H MORE EVALUATION METRICS OF EXPERIMENTS ON EBAY-LARGE

H.1 True positive rate (TPR)/Recall, true negative rate (TNR), false positive rate (FPR), false negative rate (FNR)

We document the TPR, TNR, FPR, and FNR on the test set of *eBay-xlarge* in Tables 14, 15, and 16. Note that FPR = 1 - TNR and FNR = 1 - TPR. TPR is also known as recall.

H.2 Precision and recall at various thresholds

In Tables 17, 18, and 19, we list the precision and recall scores calculated at various thresholds on the test set of *eBay-xlarge*.

H.3 *ROC curve on eBay-large.*

Besides the ROC curve in Figure 9 showing FPR <0.1 in flagging frauds on *eBay-large*, we also present the ROC curves in the full range of FPR. The xFraud detector+ outperforms all baselines in ROC-AUC in the two experiments on 8 and 16 machines, respectively.

H.4 Discussion: Performance analysis of xFraud in production

Reading from Tables 18 and 19, on the whole test set of *eBay-large* with 4.33% fraud rate, our method

- with threshold 0.983 has precision = 0.9822 and recall = 0.1091,
- with threshold 0.977 has precision = 0.9539 and recall = 0.2063,
- with threshold 0.960 has precision = 0.9217 and recall = 0.2930.

As we discussed in Appendix B, we filtered and sampled the transaction labels in the *eBay-xlarge* dataset before applying GNN models. These operations lead to the change of fraud rate in the transaction labels in each step:

$$\begin{array}{c}\text{(1) the original data stream (0.016\% fraud)}\\ \text{[filtered by rules]}\Big\Downarrow\\ \text{(2) the filtered data stream (0.043\% fraud)}\\ \text{[sampled all frauds \& 10\% benign]}\Big\Downarrow\\ \text{(3) the sampled data stream (4.33\% fraud)}\ .\end{array}$$

In this paper, we report all numbers on step (3). *How would the algorithm perform on the dataset on step (2)?* We further report precision on step (2). In this case, a 0.98 precision on (3) corresponds to 0.32 precision on (2), with 0.1 recall; and a 0.95 precision on (3) corresponds to 0.16 precision on (2), with 0.2 recall.

A 0.32 precision means that for 3 fraud candidates investigated by the business unit, 1 will be a real fraud. This, with 0.1 recall is acceptable in our scenario. Even a 0.16 precision, with 0.2 recall, is reasonable – for every 6 fraud candidates investigated by the business unit, 1 will be a real fraud.

When our model is deployed into production, there will be more considerations beyond this single GNN model — one has to consider the entire pipeline, including data preprocessing, feature engineering, and downsampling. For example, in many eBay applications, less risky transactions are filtered out by rule-based or ML-based strategies before using more complicated models such as GNN; and GEM [28] has also pre-filtered isolated transactions.

H.5 Potential production scenario using xFraud

We have used an industrial-scale dataset *eBay-xlarge* using seven months of transaction data produced at eBay. xFraud now has a model trained on historical data. What is also practical is an incremental setting of online model training and fine-tuning. For example, we can perform model updates on a daily basis to ensure the model timeliness. We can also build models in an incremental setting. For instance, we use the data from the $T-1$ week (or month) to flag the transactions produced in the T week (or month). However, there is a caveat here: many fraudulent behaviors are planned for the long-term attacks. The defaulters would "cultivate" a set of accounts for many months to gain the trust of the platform and then launch attacks. Hence, it is crucial to use both historical and up-to-date data to train our system and combine their predictions in production. After our delicate optimization for the distributed xFraud system, xFraud now can efficiently train a GNN model on a billion-scale graph dataset (38s per epoch on *eBay-xlarge* using 16 machines). With xFraud's powerful processing capabilities, we can train a new model within several hours in eBay, and hence support real production scenarios. Besides, xFraud can also accelerate the inference task significantly in a distributed setting (less than 0.1 seconds to process a batch of 640 nodes using a single machine). To help downstream tasks in a production scenario, the inference scores for the transactions are attached to the corresponding entities, represented as risk scores for the upcoming transaction events. These scores are treated as features for downstream risk models and as variables for the rule-based defense systems.

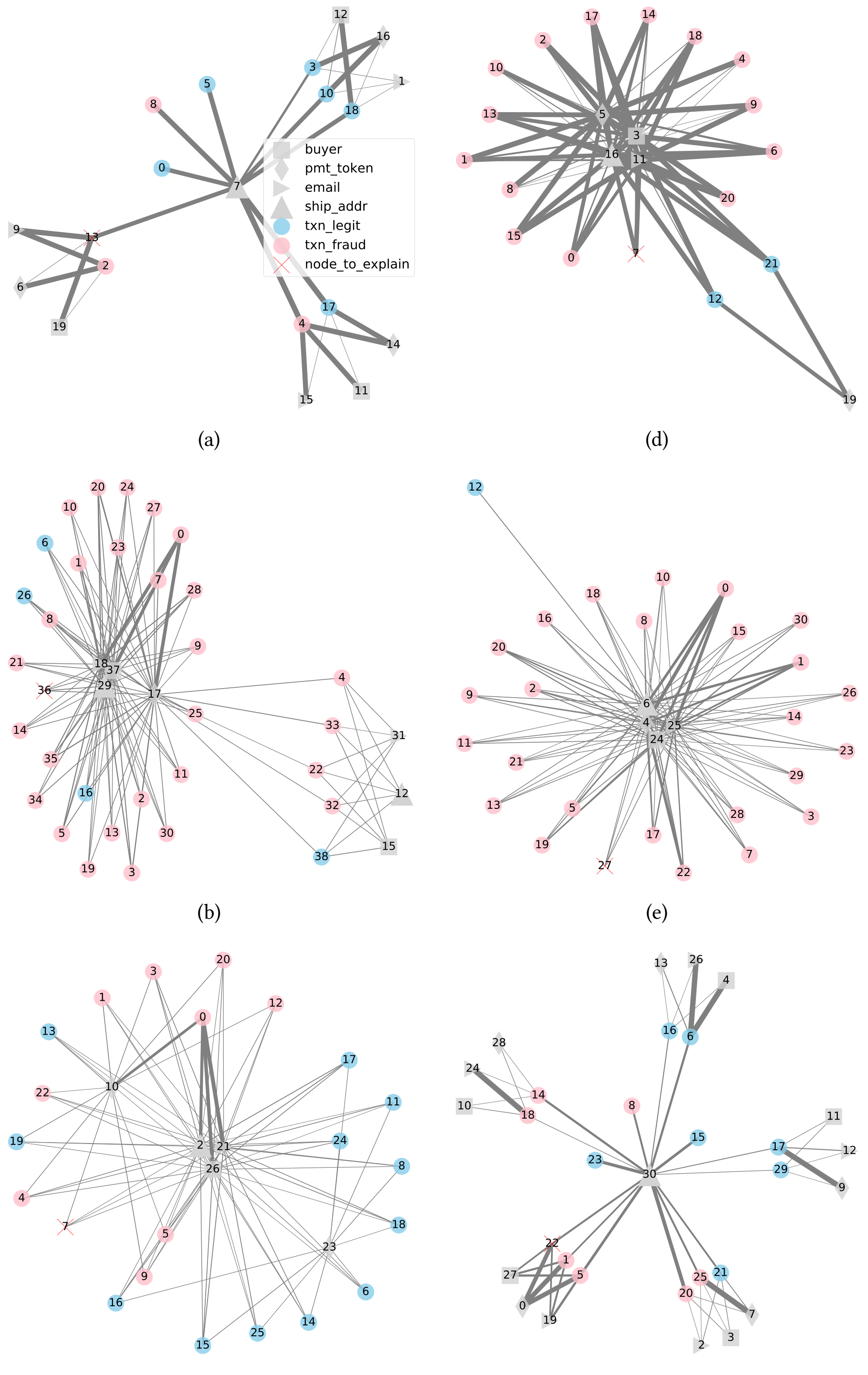
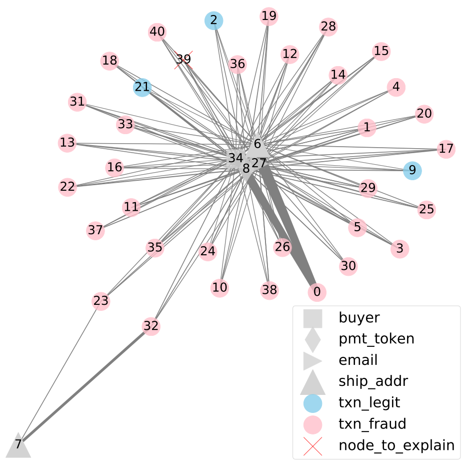
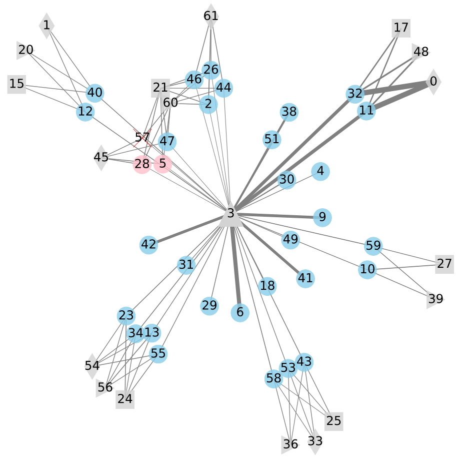


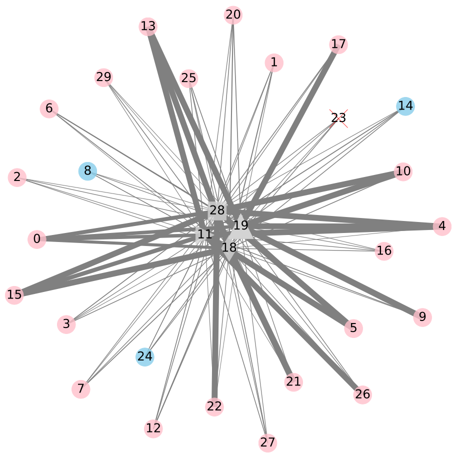
Figure 16: Case studies using hybrid learner weights: true positives (TP).



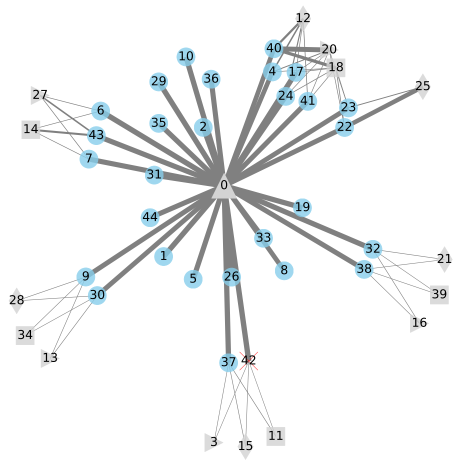
(a) FP: benign ⟶ fraudulent (case 1)



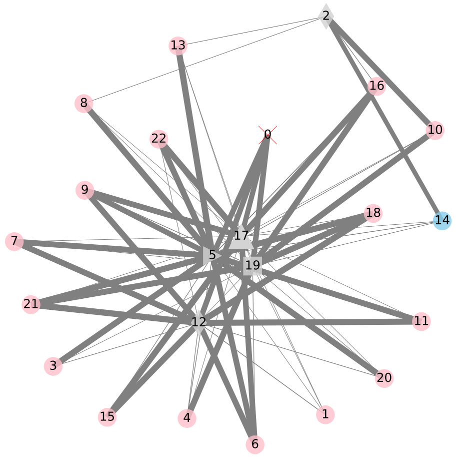
(d) FN: fraudulent ⟶ benign (case 1)



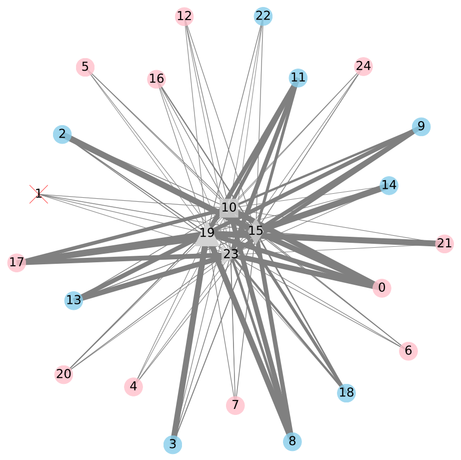
(b) FP: benign ⟶ fraudulent (case 2)



(e) FN: fraudulent ⟶ benign (case 2)



(c) FP: benign ⟶ fraudulent (case 3)



(f) FN: fraudulent ⟶ benign (case 3)

Figure 17: Case studies using hybrid learner weights: false positives (a,b,c on the left) and false negatives (d,e,f on the right).

Table 14: TPR, FNR, FPR, and TNR on *eBay-xlarge* by varying thresholds from 0.1 to 0.9 on the prediction scores.
FNR = 1 - TPR. The higher TPR is, the better; the lower FNR, the better.
FPR = 1 - TNR. The higher TNR is, the better; the lower FPR, the better.

Model	**# of machines**	**Seed**	**TPR@threshold**								
			0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
GAT	8	A	0.7379	0.6718	0.6264	0.5891	0.5535	0.5158	0.4730	0.4187	0.3170
		B	0.7443	0.6664	0.6184	0.5787	0.5391	0.5030	0.4577	0.3972	0.2780
	16	A	0.7439	0.6789	0.6311	0.5923	0.5483	0.5112	0.4614	0.3790	0.2152
		B	0.7137	0.6287	0.5839	0.5396	0.4930	0.4345	0.3488	0.2262	0.0432
GEM	8	A	0.7125	0.6028	0.5182	0.4608	0.4058	0.3563	0.3086	0.2501	0.1599
		B	0.6737	0.5611	0.4913	0.4363	0.3788	0.3349	0.2902	0.2344	0.1360
	16	A	0.7052	0.6061	0.5263	0.4592	0.4073	0.3490	0.2859	0.2014	0.0660
		B	0.6649	0.5646	0.4955	0.4331	0.3829	0.3282	0.2522	0.1742	0.0588
xFraud detector+	8	A	0.5731	0.5083	0.4794	0.4582	0.4407	0.4268	0.4103	0.3933	0.3621
		B	0.5480	0.4801	0.4482	0.4213	0.4067	0.3904	0.3746	0.3579	0.3199
	16	A	0.5056	0.4521	0.4198	0.3978	0.3757	0.3526	0.3228	0.2794	0.1424
		B	0.5212	0.4549	0.4177	0.3902	0.3524	0.3113	0.2529	0.1597	0.0018

Model	**# of machines**	**Seed**	**FNR@threshold**								
			0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
GAT	8	A	0.2621	0.3282	0.3736	0.4109	0.4465	0.4842	0.5270	0.5813	0.6830
		B	0.2557	0.3336	0.3816	0.4213	0.4609	0.4970	0.5423	0.6028	0.7220
	16	A	0.2561	0.3211	0.3689	0.4077	0.4517	0.4888	0.5386	0.6210	0.7848
		B	0.2863	0.3713	0.4161	0.4604	0.5070	0.5655	0.6512	0.7738	0.9568
GEM	8	A	0.2875	0.3972	0.4818	0.5392	0.5942	0.6437	0.6914	0.7499	0.8401
		B	0.3263	0.4389	0.5087	0.5637	0.6212	0.6651	0.7098	0.7656	0.8640
	16	A	0.2948	0.3939	0.4737	0.5408	0.5927	0.6510	0.7141	0.7986	0.9340
		B	0.3351	0.4354	0.5045	0.5669	0.6171	0.6718	0.7478	0.8258	0.9412
xFraud detector+	8	A	0.4269	0.4917	0.5206	0.5418	0.5593	0.5732	0.5897	0.6067	0.6379
		B	0.4520	0.5199	0.5518	0.5787	0.5933	0.6096	0.6254	0.6421	0.6801
	16	A	0.4944	0.5479	0.5802	0.6022	0.6243	0.6474	0.6772	0.7206	0.8576
		B	0.4788	0.5451	0.5823	0.6098	0.6476	0.6887	0.7471	0.8403	0.9982

Model	**# of machines**	**Seed**	**TNR@threshold**								
			0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
GAT	8	A	0.8673	0.9077	0.9268	0.9400	0.9506	0.9597	0.9688	0.9777	0.9905
		B	0.8604	0.9041	0.9246	0.9380	0.9486	0.9586	0.9678	0.9779	0.9907
	16	A	0.8650	0.9057	0.9265	0.9403	0.9523	0.9631	0.9734	0.9837	0.9961
		B	0.8792	0.9166	0.9358	0.9501	0.9628	0.9730	0.9829	0.9933	0.9995
GEM	8	A	0.9019	0.9440	0.9621	0.9729	0.9802	0.9859	0.9903	0.9945	0.9981
		B	0.9189	0.9539	0.9697	0.9787	0.9840	0.9882	0.9924	0.9959	0.9989
	16	A	0.8987	0.9411	0.9600	0.9716	0.9799	0.9870	0.9918	0.9963	0.9995
		B	0.9134	0.9497	0.9662	0.9759	0.9834	0.9890	0.9934	0.9973	0.9997
xFraud detector+	8	A	0.9721	0.9856	0.9902	0.9927	0.9941	0.9952	0.9961	0.9969	0.9978
		B	0.9764	0.9888	0.9927	0.9945	0.9958	0.9967	0.9974	0.9980	0.9986
	16	A	0.9768	0.9882	0.9920	0.9946	0.9962	0.9973	0.9980	0.9989	0.9998
		B	0.9672	0.9805	0.9863	0.9899	0.9927	0.9952	0.9975	0.9991	1.0000

Model	**# of machines**	**Seed**	**FPR@threshold**								
			0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
GAT	8	A	0.1327	0.0923	0.0732	0.0600	0.0494	0.0403	0.0312	0.0223	0.0095
		B	0.1396	0.0959	0.0754	0.0620	0.0514	0.0414	0.0322	0.0221	0.0093
	16	A	0.1350	0.0943	0.0735	0.0597	0.0477	0.0369	0.0266	0.0163	0.0039
		B	0.1208	0.0834	0.0642	0.0499	0.0372	0.0270	0.0171	0.0067	0.0005
GEM	8	A	0.0981	0.0560	0.0379	0.0271	0.0198	0.0141	0.0097	0.0055	0.0019
		B	0.0811	0.0461	0.0303	0.0213	0.0160	0.0118	0.0076	0.0041	0.0011
	16	A	0.1013	0.0589	0.0400	0.0284	0.0201	0.0130	0.0082	0.0037	0.0005
		B	0.0866	0.0503	0.0338	0.0241	0.0166	0.0110	0.0066	0.0027	0.0003
xFraud detector+	8	A	0.0279	0.0144	0.0098	0.0073	0.0059	0.0048	0.0039	0.0031	0.0022
		B	0.0236	0.0112	0.0073	0.0055	0.0042	0.0033	0.0026	0.0020	0.0014
	16	A	0.0232	0.0118	0.0080	0.0054	0.0038	0.0027	0.0020	0.0011	0.0002
		B	0.0328	0.0195	0.0137	0.0101	0.0073	0.0048	0.0025	0.0009	0.0000

Table 15: TPR, FNR, FPR, and TNR on *eBay-xlarge* by varying thresholds from 0.95 to 0.977 on the prediction scores.
FNR = 1 - TPR. The higher TPR is, the better; the lower FNR, the better.
FPR = 1 - TNR. The higher TNR is, the better; the lower FPR, the better.
"-" denotes that the scores do not exist for scores ⩾ threshold.

Model	**# of machines**	**Seed**	**TPR@threshold**									
			0.95	**0.96**	**0.97**	**0.971**	**0.972**	**0.973**	**0.974**	**0.975**	**0.976**	**0.977**
GAT	8	A	0.1820	0.1303	0.0717	0.0640	0.0595	0.0541	0.0471	0.0412	0.0362	0.0319
		B	0.1408	0.0903	0.0411	0.0363	0.0326	0.0285	0.0234	0.0188	0.0153	0.0123
	16	A	0.0380	0.0065	-	-	-	-	-	-	-	-
		B	0.0004	-	-	-	-	-	-	-	-	-
GEM	8	A	0.0699	0.0477	0.0192	0.0168	0.0142	0.0131	0.0113	0.0093	0.0080	0.0057
		B	0.0461	0.0171	0.0024	0.0020	0.0014	0.0014	0.0012	0.0010	0.0008	-
	16	A	0.0028	0.0008	-	-	-	-	-	-	-	-
		B	0.0006	-	-	-	-	-	-	-	-	-
xFraud detector+	8	A	0.3164	0.2930	0.2634	0.2548	0.2482	0.2401	0.2342	0.2255	0.2155	0.2063
		B	0.2714	0.2484	0.2024	0.1940	0.1867	0.1772	0.1663	0.1527	0.1381	0.1255
	16	A	0.0111	0.0014	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-

Model	**# of machines**	**Seed**	**FNR@threshold**									
			0.95	**0.96**	**0.97**	**0.971**	**0.972**	**0.973**	**0.974**	**0.975**	**0.976**	**0.977**
GAT	8	A	0.8180	0.8697	0.9283	0.9360	0.9405	0.9459	0.9529	0.9588	0.9638	0.9681
		B	0.8592	0.9097	0.9589	0.9637	0.9674	0.9715	0.9766	0.9812	0.9847	0.9877
	16	A	0.9620	0.9935	-	-	-	-	-	-	-	-
		B	0.9996	-	-	-	-	-	-	-	-	-
GEM	8	A	0.9301	0.9523	0.9808	0.9832	0.9858	0.9869	0.9887	0.9907	0.9920	0.9943
		B	0.9539	0.9829	0.9976	0.9980	0.9986	0.9986	0.9988	0.9990	0.9992	-
	16	A	0.9972	0.9992	-	-	-	-	-	-	-	-
		B	0.9994	-	-	-	-	-	-	-	-	-
xFraud detector+	8	A	0.6836	0.7070	0.7366	0.7452	0.7518	0.7599	0.7658	0.7745	0.7845	0.7937
		B	0.7286	0.7516	0.7976	0.8060	0.8133	0.8228	0.8337	0.8473	0.8619	0.8745
	16	A	0.9889	0.9986	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-

Model	**# of machines**	**Seed**	**TNR@threshold**									
			0.95	**0.96**	**0.97**	**0.971**	**0.972**	**0.973**	**0.974**	**0.975**	**0.976**	**0.977**
GAT	8	A	0.9977	0.9991	0.9998	0.9999	0.9999	0.9999	0.9999	0.9999	1.0000	1.0000
		B	0.9980	0.9992	0.9998	0.9998	0.9999	0.9999	0.9999	0.9999	1.0000	1.0000
	16	A	1.0000	1.0000	-	-	-	-	-	-	-	-
		B	1.0000	-	-	-	-	-	-	-	-	-
GEM	8	A	0.9996	0.9999	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
		B	0.9999	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	-
	16	A	1.0000	1.0000	-	-	-	-	-	-	-	-
		B	1.0000	-	-	-	-	-	-	-	-	-
xFraud detector+	8	A	0.9986	0.9989	0.9992	0.9992	0.9992	0.9993	0.9994	0.9994	0.9995	0.9995
		B	0.9991	0.9993	0.9996	0.9997	0.9998	0.9998	0.9998	0.9999	0.9999	0.9999
	16	A	1.0000	1.0000	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-

Model	**# of machines**	**Seed**	**FPR@threshold**									
			0.95	**0.96**	**0.97**	**0.971**	**0.972**	**0.973**	**0.974**	**0.975**	**0.976**	**0.977**
GAT	8	A	0.0023	0.0009	0.0002	0.0001	0.0001	0.0001	0.0001	0.0001	0.0000	0.0000
		B	0.0020	0.0008	0.0002	0.0002	0.0001	0.0001	0.0001	0.0001	0.0000	0.0000
	16	A	0.0000	0.0000	-	-	-	-	-	-	-	-
		B	0.0000	-	-	-	-	-	-	-	-	-
GEM	8	A	0.0004	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
		B	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	-
	16	A	0.0000	0.0000	-	-	-	-	-	-	-	-
		B	0.0000	-	-	-	-	-	-	-	-	-
xFraud detector+	8	A	0.0014	0.0011	0.0008	0.0008	0.0008	0.0007	0.0006	0.0006	0.0005	0.0005
		B	0.0009	0.0007	0.0004	0.0003	0.0002	0.0002	0.0002	0.0001	0.0001	0.0001
	16	A	0.0000	0.0000	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-

Table 16: TPR, FNR, FPR, and TNR on *eBay-xlarge* by varying thresholds from 0.978 to 0.987 on the prediction scores.
FNR = 1 - TPR. The higher TPR is, the better; the lower FNR, the better.
FPR = 1 - TNR. The higher TNR is, the better; the lower FPR, the better.
“-" denotes that the scores do not exist for scores ⩾ threshold.

Model	**# of machines**	**Seed**	**TPR@threshold**									
			0.978	**0.979**	**0.98**	**0.981**	**0.982**	**0.983**	**0.984**	**0.985**	**0.986**	**0.987**
GAT	8	A	0.0265	0.0181	0.0130	0.0088	0.0038	0.0008	0.0002	-	-	-
		B	0.0095	0.0056	0.0036	0.0028	0.0018	0.0006	-	-	-	
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
GEM	8	A	0.0045	0.0027	0.0018	0.0005	0.0005	0.0004	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
xFraud detector+	8	A	0.1945	0.1820	0.1655	0.1504	0.1312	0.1091	0.0842	0.0623	0.0399	0.0196
		B	0.1102	0.0959	0.0823	0.0642	0.0443	0.0279	0.0099	0.0037	0.0010	0.0002
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-

Model	**# of machines**	**Seed**	**FNR@threshold**									
			0.978	**0.979**	**0.98**	**0.981**	**0.982**	**0.983**	**0.984**	**0.985**	**0.986**	**0.987**
GAT	8	A	0.9735	0.9819	0.9870	0.9912	0.9962	0.9992	0.9998	-	-	-
		B	0.9905	0.9944	0.9964	0.9972	0.9982	0.9994		-	-	-
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
GEM	8	A	0.9955	0.9973	0.9982	0.9995	0.9995	0.9996	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
xFraud detector+	8	A	0.8055	0.8180	0.8345	0.8496	0.8688	0.8909	0.9158	0.9377	0.9601	0.9804
		B	0.8898	0.9041	0.9177	0.9358	0.9557	0.9721	0.9901	0.9963	0.9990	0.9998
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-

Model	**# of machines**	**Seed**	**TNR@threshold**									
			0.978	**0.979**	**0.98**	**0.981**	**0.982**	**0.983**	**0.984**	**0.985**	**0.986**	**0.987**
GAT	8	A	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	-	-	-
		B	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	-	-	-	-
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
GEM	8	A	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
xFraud detector+	8	A	0.9996	0.9996	0.9997	0.9997	0.9999	0.9999	1.0000	1.0000	1.0000	1.0000
		B	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-

Model	**# of machines**	**Seed**	**FPR@threshold**									
			0.978	**0.979**	**0.98**	**0.981**	**0.982**	**0.983**	**0.984**	**0.985**	**0.986**	**0.987**
GAT	8	A	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	-	-	-
		B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	-	-	-	-
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
GEM	8	A	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
xFraud detector+	8	A	0.0004	0.0004	0.0003	0.0003	0.0001	0.0001	0.0000	0.0000	0.0000	0.0000
		B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-

Table 17: Precision and recall on *eBay-xlarge* by varying thresholds from 0.1 to 0.9 on the prediction scores.

Model	# of machines	Seed	Precision@threshold								
			0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
GAT	8	A	0.2010	0.2478	0.2792	0.3074	0.3362	0.3666	0.4071	0.4592	0.6021
		B	0.1943	0.2391	0.2706	0.2968	0.3219	0.3544	0.3917	0.4488	0.5755
	16	A	0.1996	0.2456	0.2797	0.3098	0.3420	0.3852	0.4397	0.5124	0.7124
		B	0.2109	0.2542	0.2916	0.3285	0.3750	0.4217	0.4796	0.6028	0.8057
GEM	8	A	0.2472	0.3276	0.3822	0.4351	0.4812	0.5329	0.5908	0.6741	0.7904
		B	0.2731	0.3553	0.4231	0.4814	0.5175	0.5618	0.6321	0.7213	0.8509
	16	A	0.2395	0.3178	0.3729	0.4228	0.4788	0.5488	0.6133	0.7118	0.8525
		B	0.2579	0.3369	0.3985	0.4481	0.5105	0.5737	0.6352	0.7460	0.9040
xFraud detector+	8	A	0.4820	0.6144	0.6894	0.7383	0.7713	0.8025	0.8249	0.8529	0.8798
		B	0.5122	0.6588	0.7346	0.7771	0.8157	0.8409	0.8660	0.8924	0.9119
	16	A	0.4960	0.6345	0.7030	0.7679	0.8188	0.8552	0.8791	0.9223	0.9689
		B	0.4179	0.5133	0.5805	0.6363	0.6871	0.7466	0.8188	0.8939	0.8846

Model	# of machines	Seed	Recall@threshold								
			0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
GAT	8	A	0.7379	0.6718	0.6264	0.5891	0.5535	0.5158	0.4730	0.4187	0.3170
		B	0.7443	0.6664	0.6184	0.5787	0.5391	0.5030	0.4577	0.3972	0.2780
	16	A	0.7439	0.6789	0.6311	0.5923	0.5483	0.5112	0.4614	0.3790	0.2152
		B	0.7137	0.6287	0.5839	0.5396	0.4930	0.4345	0.3488	0.2262	0.0432
GEM	8	A	0.7125	0.6028	0.5182	0.4608	0.4058	0.3563	0.3086	0.2501	0.1599
		B	0.6737	0.5611	0.4913	0.4363	0.3788	0.3349	0.2902	0.2344	0.1360
	16	A	0.7052	0.6061	0.5263	0.4592	0.4073	0.3490	0.2859	0.2014	0.0660
		B	0.6649	0.5646	0.4955	0.4331	0.3829	0.3282	0.2522	0.1742	0.0588
xFraud detector+	8	A	0.5731	0.5083	0.4794	0.4582	0.4407	0.4268	0.4103	0.3933	0.3621
		B	0.5480	0.4801	0.4482	0.4213	0.4067	0.3904	0.3746	0.3579	0.3199
	16	A	0.5056	0.4521	0.4198	0.3978	0.3757	0.3526	0.3228	0.2794	0.1424
		B	0.5212	0.4549	0.4177	0.3902	0.3524	0.3113	0.2529	0.1597	0.0018

Table 18: Precision and recall on *eBay-xlarge* by varying thresholds from 0.95 to 0.977 on the prediction scores. "-" denotes that the scores do not exist for scores ⩾ threshold.

Model	# of machines	Seed	Precision@threshold									
			0.95	**0.96**	**0.97**	**0.971**	**0.972**	**0.973**	**0.974**	**0.975**	**0.976**	**0.977**
GAT	8	A	0.7832	0.8682	0.9392	0.9568	0.9606	0.9634	0.9642	0.9661	0.9814	0.9882
		B	0.7647	0.8358	0.9000	0.9034	0.9224	0.9144	0.9331	0.9251	0.9481	0.9759
	16	A	0.9784	1.0000	-	-	-	-	-	-	-	-
		B	1.0000	-	-	-	-	-	-	-	-	-
GEM	8	A	0.8930	0.9372	0.9655	0.9607	0.9541	0.9503	0.9430	0.9313	0.9813	1.0000
		B	0.9712	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	-
	16	A	0.9024	1.0000	-	-	-	-	-	-	-	-
		B	1.0000	-	-	-	-	-	-	-	-	-
xFraud detector+	8	A	0.9106	0.9217	0.9359	0.9370	0.9373	0.9412	0.9436	0.9433	0.9500	0.9539
		B	0.9313	0.9392	0.9613	0.9637	0.9722	0.9736	0.9785	0.9795	0.9837	0.9868
	16	A	0.9733	1.0000	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-

Model	# of machines	Seed	Recall@threshold									
			0.95	**0.96**	**0.97**	**0.971**	**0.972**	**0.973**	**0.974**	**0.975**	**0.976**	**0.977**
GAT	8	A	0.1820	0.1303	0.0717	0.0640	0.0595	0.0541	0.0471	0.0412	0.0362	0.0319
		B	0.1408	0.0903	0.0411	0.0363	0.0326	0.0285	0.0234	0.0188	0.0153	0.0123
	16	A	0.0380	0.0065	-	-	-	-	-	-	-	-
		B	0.0004	-	-	-	-	-	-	-	-	-
GEM	8	A	0.0699	0.0477	0.0192	0.0168	0.0142	0.0131	0.0113	0.0093	0.0080	0.0057
		B	0.0461	0.0171	0.0024	0.0020	0.0014	0.0014	0.0012	0.0010	0.0008	-
	16	A	0.0028	0.0008	-	-	-	-	-	-	-	-
		B	0.0006	-	-	-	-	-	-	-	-	-
xFraud detector+	8	A	0.3164	0.2930	0.2634	0.2548	0.2482	0.2401	0.2342	0.2255	0.2155	0.2063
		B	0.2714	0.2484	0.2024	0.1940	0.1867	0.1772	0.1663	0.1527	0.1381	0.1255
	16	A	0.0111	0.0014	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-

Table 19: Precision and recall on *eBay-xlarge* by varying thresholds from 0.978 to 0.987 on the prediction scores. “-" denotes that the scores do not exist for scores ⩾ threshold.

Model	# of machines	Seed	Precision@threshold									
			0.978	**0.979**	**0.98**	**0.981**	**0.982**	**0.983**	**0.984**	**0.985**	**0.986**	**0.987**
GAT	8	A	0.9971	0.9958	0.9942	1.0000	1.0000	1.0000	1.0000	-	-	-
		B	0.9690	0.9487	0.9216	0.9024	0.8519	0.6667	-	-	-	-
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
GEM	8	A	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
xFraud detector+	8	A	0.9537	0.9545	0.9586	0.9625	0.9829	0.9822	0.9910	0.9879	0.9905	0.9809
		B	0.9918	0.9906	0.9982	0.9976	0.9966	1.0000	1.0000	1.0000	1.0000	1.0000
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-

Model	# of machines	Seed	Recall@threshold									
			0.978	**0.979**	**0.98**	**0.981**	**0.982**	**0.983**	**0.984**	**0.985**	**0.986**	**0.987**
GAT	8	A	0.0265	0.0181	0.0130	0.0088	0.0038	0.0008	0.0002	-	-	-
		B	0.0095	0.0056	0.0036	0.0028	0.0018	0.0006	-	-	-	-
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
GEM	8	A	0.0045	0.0027	0.0018	0.0005	0.0005	0.0004	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-
16	A	-	-	-	-	-	-	-	-	-	-	
		B	-	-	-	-	-	-	-	-	-	-
xFraud detector+	8	A	0.1945	0.1820	0.1655	0.1504	0.1312	0.1091	0.0842	0.0623	0.0399	0.0196
		B	0.1102	0.0959	0.0823	0.0642	0.0443	0.0279	0.0099	0.0037	0.0010	0.0002
	16	A	-	-	-	-	-	-	-	-	-	-
		B	-	-	-	-	-	-	-	-	-	-