International Journal of Sustainability and Innovation in Engineering (IJSIE)
https://www.doi.org/10.56830/IJSIE2026

Vol.4 No.1

Adaptive Data Pipeline Architectures for Evolving Fraud Patterns Using Graph ML

https://www.doi.org/10.56830/IJSIE202603

Dharam Pal Singh
Lead Architect (Data Platform)/Mphasis
dharam.singh@mphasis.com

Received: 3th Jan, 2025, Accepted: 5th March, 2025, Published: 15th March, 2026

Abstract

The growing sophistication of financial fraud has intensified alongside digital transformation, real-time payment systems, and the expansion of online financial services, resulting in substantial financial losses and operational challenges. Organizations combat these evolving threats by deploying advanced analytics and machine learning models capable of detecting unusual patterns across large volumes of transactional and behavioral data. Yet this approach creates tension: the financial data required for effective fraud detection—particularly personally identifiable information and transaction records—is inherently sensitive and subject to significant privacy protections.

This paper introduces a comprehensive framework for integrating machine learning into ETL pipelines while preserving privacy, enabling real-time financial fraud detection that is both data-driven and secure. The architecture embeds privacy, security, and regulatory compliance throughout every pipeline stage—from initial data ingestion and transformation through model training to real-time fraud alert generation. Our approach combines multiple privacy-enhancing technologies, including differential privacy, homomorphic encryption, federated learning, and secure multi-party computation, allowing organizations to perform collaborative analytics without exposing raw or sensitive data to unauthorized parties. The system incorporates temporal and behavioral modeling alongside external data enrichment and automated fraud registry capabilities, enhancing its ability to identify sophisticated fraud patterns as they evolve. Pipeline orchestration ensures scalability and near real-time processing, delivering timely fraud risk assessments. Experimental results indicate substantial improvements in both detection accuracy and processing speed relative to conventional approaches. Performance gains are further amplified through dimensionality reduction techniques. This framework enables data processing systems to scale dynamically in response to changing demands while preserving operational efficiency and resilience. The resulting ML-enhanced ETL pipeline equips financial institutions with an effective mechanism for minimizing fraud losses while maintaining both operational agility and regulatory compliance.

Keyword: Fraud detection, graph machine learning, graph neural networks, adaptive systems, streaming data pipelines, concept drift, continual learning.

1. Introduction

1.1. The Evolution of Fraud and Detection Challenges

Around the world, governments and banks have been making strong headway in the efforts to stop the threat of scams. However, new research from the Global Anti-Scam Alliance shows consumers from 42 countries lost roughly $442 billion to scams in the past year. This means there is still much work to be done to protect people from scams' financial and emotional pain. GASA's report surveyed 46,000 adults worldwide on their experiences with scams. The findings, outlined that scams are a persistent global threat, with roughly half of adults impacted by one at some time. On a global scale, scams are becoming increasingly common. GASA's research found that 57% of respondents were scammed in the past year. GASA's data shows that at least a simple majority of respondents have encountered a scam in the last 12 months in seven global markets. Scam activity is far more common in South America and Oceania, where 72% and 73% of residents, respectively, have experienced one. In response, banks have begun relying on advanced analytics and machine learning (ML) to identify fraudulent behavior embedded within vast amounts of transactional data that streams by the second. However, the success of these analytics business systems hinges on the existence of accurate, timely, and complete data, which is likely to include personally identifiable and financial attributes.

1.2. Graph Machine Learning for Fraud Detection

Graph machine learning is used for fraud detection by analyzing the connections and relationships between entities in a network (Finn, Abbeel, & Levine, 2017). It can be applied to a wide range of datasets, such as financial transactions, communication logs, and social media interactions, to uncover patterns and anomalies that indicate fraudulent activity.

One common approach is to construct a graph based on the relationships between entities, such as financial accounts, individuals, or IP addresses. This graph can then be analyzed to identify groups of entities that exhibit suspicious behavior, such as abnormal patterns of communication or financial transactions.

The typical process of getting to prepare a graph ML model for use in fraud detection can be divided into several phases: data collection and cleaning, defining entities and relationships, building the graph, training the graph neural network (GNN) on the constructed graph, and deploying the model for inference.

1.3. Concept Drift in Fraud Detection

Concept drift refers to the change in the statistical properties of input data over time, which causes machine learning models to lose predictive accuracy.

In the compliance context, concept drift means that models trained on historical transaction data may no longer recognize suspicious behavior if criminal strategies evolve. For example, a fraud detection model may initially identify unusual card spending patterns but miss new fraud tactics that emerge later. Similarly, an AML monitoring system could misclassify transactions as normal because the data distribution has shifted since the model was last trained.

This makes concept drift a significant risk factor for AML screening, regulatory compliance, and fraud prevention programs. It highlights why regulators increasingly demand model governance, explainability, and oversight in AI-driven compliance systems.

2. Adaptive Data Pipeline Architectures

2.1. Real-time Streaming Frameworks

Streaming data pipelines are data pipelines that are designed to ingest, process, and transport data continuously from one or multiple sources in real-time. They run continuously as compared to batch pipelines, which process the data at fixed intervals.

Streaming data pipelines handle the data as it arrives thereby enabling quick analysis and decision-making for businesses. They is crucial where timing is critical, such as monitoring work, traffic data, transaction data to avoid fraudulent activity, sensor data, or personalized experience in real-time.

The various streaming tools are Apache Kafka, Apache Flink, AWS Kinesis, GCP Pub-Sub, etc.

Real-time fraud detection is critical for any financial service to safeguard customer's assets. The Streaming data pipeline along with machine learning analyses the transaction patterns and flags anomalies. Streaming data pipelines analyze streams from multiple sources (e.g. transaction logs) to detect fraud activity.

Example: Streaming data pipeline can monitor the users transactions for unusual activity, such as sudden purchases, gambling purchases, frequent transactions at a location, or any other activity that may be treated as fraudulent (Kumar, Hooi, Makhija, Kumar, Faloutsos, & Subrahmanian, 2018). Streaming data pipeline triggers the alerts that can block the transactions and alert the users to take preventive actions.

2.2. Feature Engineering and Enrichment

Key Features of Streaming Data Pipelines

Real-Time Processing: Streaming data pipelines run continuously to ingest the data as they arrive, analyze it using a set of logic, and generate results for consumption.

Scalability: They should have the ability to handle high data volumes without compromising performance. Tools like Apache Kafka, Apache Flink, AWS Kinesis, and GCP Pub-Sub are a few examples that handle scalability without impacting performance.

Low Latency: They offer low latency so that insights and decisions can be made almost instantly. This is a very crucial feature of any streaming data pipeline as delay in the analysis can cost heavily.

Flexibility: It supports diverse use cases, from IoT applications to fraud detection.

For example, a taxi service like Uber, OLA, Left uses streaming pipelines to match the riders and drivers' position, estimate the arrival and departure times, and provide live traffic updates. These services use the streaming data pipeline in the background to ensure seamless, real-time experiences. Read the advantages of data pipeline and leverage its benefits to the fullest.

There are different stages as per the requirements, however, the streaming data pipelines has four major stages as described below –

Data Ingestion: It is the initial step in the streaming data pipeline. This step involves collecting raw data from multiple sources in real time and ingesting them for further analysis.

The various sources like IoT devices, application logs, user interaction, financial transactions, social media posts, etc., generate a high volume of real-time data.

Tools like Apache Kafka, Amazon Kinesis, Google Pub Sub, and RabbitMQ are commonly used for high-throughput real-time data ingestion.

Example: Social media platforms like Twitter, Facebook, Instagram, etc. use ingestion pipelines to process millions of tweets/events per second.

Data Processing: When the data is ingested to the system, it is then processed to derive meaningful insights from the data in the real-time. Streaming data pipeline involves the following steps for data processing :

At first, the data is filtered to remove duplicates, junk records, and other irrelevant information. Then the aggregation is performed on the data to gather the insights by applying business rules. The data is then enriched with additional data sources to increase the quality of the data. Incoming data streams are processed in real-time using frameworks like Apache Flink, Spark Streaming, or Google DataFlow.

Data Storage: Once the data is processed, it is stored in the optimized format for quick retrieval, thereby reducing latency. The popular databases used for storage are ElasticSearch, Redis, Snowflake, etc.

Example: A stock trading application or real-time sports streaming uses a high-performance database to store real-time market data for instant access.

Data Delivery: Data delivery is the last step of the pipeline. This step makes the analyzed and processed data available for the end user or any downstream application. The data is available via API, dashboards, or real-time reports.

2.3. Common Data Pipeline Architecture Patterns

While the components are the building blocks, architectural patterns are the blueprints. Choosing the right pattern is critical and depends entirely on your specific requirements for latency, scalability, data volume, complexity, and cost. Here are some of the most common pipeline blueprints used today.

Lambda Architecture

A popular but complex pattern, Lambda architecture attempts to provide a balance between real-time speed and batch-processing reliability. It does this by running parallel data flows: a “hot path” (speed layer) for real-time streaming data and a “cold path” (batch layer) for comprehensive, historical batch processing. The results are then merged in a serving layer.

Best for: Use cases that need both low-latency, real-time views and highly accurate, comprehensive historical reporting.

Challenge: It introduces significant complexity, requiring teams to maintain two separate codebases and processing systems, which can be costly and difficult to manage.

Kappa Architecture

Kappa architecture emerged as a simpler alternative to Lambda. It eliminates the batch layer entirely and handles all processing—both real-time and historical—through a single streaming pipeline. Historical analysis is achieved by reprocessing the stream from the beginning.

Best for: Scenarios where most data processing can be handled in real time and the logic doesn't require a separate batch system. It's ideal for event-driven systems.

Challenge: Reprocessing large historical datasets can be computationally expensive and slow, making it less suitable for use cases requiring frequent, large-scale historical analysis (IEEE Computational Intelligence Society, 2019).

Event-Driven Architectures

This pattern decouples data producers from data consumers using an event-based model. Systems communicate by producing and consuming events (e.g., "customer_created," "order_placed") via a central messaging platform like Kafka. Each microservice can process these events independently, creating a highly scalable and resilient system.

Best for: Complex, distributed systems where agility and scalability are paramount. It's the foundation for many modern cloud-native applications.

Challenge: Can lead to complex data consistency and management challenges across dozens or even hundreds of independent services.

Hybrid and CDC-First Architectures

This pragmatic approach acknowledges that most enterprises live in a hybrid world, with data in both legacy on-premises systems and modern cloud platforms. A Change Data Capture (CDC)-first architecture focuses on efficiently capturing granular changes (inserts, updates, deletes) from source databases in real time. This data can then feed both streaming analytics applications and batch-based data warehouses simultaneously.

Best for: Organizations modernizing their infrastructure, migrating to the cloud, or needing to sync data between operational and analytical systems with minimal latency and no downtime.

Challenge: Requires specialized tools that can handle low-impact CDC from a wide variety of database sources.

3. Fundamentals of Graph-Based Fraud Detection

3.1. Graph Representations of Financial Networks

An overview of the road map of this paper. We first introduce the mainly utilized graph neural networks (GNNs) for financial fraud detection. Secondly, we summarize the reasons for choosing GNNs (Liu, et al., 2021). After that, we introduce the methods and special tricks in GNN architecture design for financial fraud detection tasks. Then we go through the real-world applications and future directions for GNN-based financial fraud detection.

In this section, we first provide the definitions of the common notations used, as shown in Table 1. We define graph G = (V, E, A) as an undirected graph, where V is the set of nodes, |V| = n is the number of nodes, E is the set of edges, and A ∈ R n×n is the adjacency matrix; the entry Ai, j of the matrix is 1 if there is an edge between nodes i and j, and 0 otherwise. The degree matrix D ∈ R n×n is a diagonal matrix where each entry Di,i is the sum of the i-th row of A, P j Ai, j . The graph Laplacian matrix is defined as L = In− D − 1 2 AD− 1 2 , where In ∈ R n×n is the identity matrix. The symmetric normalized Laplacian matrix is another form of the Laplacian matrix, defined as L = UΛU ⊤. Here U = {ui |i = 1, . . . , n} is the matrix of

eigenvectors, and $\Lambda = diag(\{\lambda i \mid i = 1, \ldots, n\})$ is the diagonal matrix of eigenvalues, and ui is the i-th eigenvector. We define $X \in R^{n \times D}$ as the node feature matrix of graph G, where $xi \in R^D$ is the feature vector of node i. Then we introduce some extended definitions as follows:

Table 1: The summary of notations in this paper

Symbol	Description
G	Graph
V	Set of nodes
E	Set of edges
n	Number of nodes
$In \in R^{n \times n}$	Identity matrix
$A \in R^{n \times n}$	Adjacency matrix
$D \in R^{n \times n}$	Degree matrix (diagonal)
$L \in R^{n \times n}$	Laplacian matrix
$U \in R^{n \times n}$	Eigenvector matrix
$\Lambda \in R^{n \times n}$	Eigenvalue matrix (diagonal)
$ui \in R^n$	i-th eigenvector
$X \in R^{n \times D}$	Node feature matrix
$xi \in R^D$	Features of the i-th node
$f \in R^n$	Signal

Homogeneous and Heterogeneous Graphs. A graph G = (V, E) can be extended to a heterogeneous graph defined as G = (V, E, ϕ, ψ), where $\phi : V \rightarrow A$ assigns types to nodes and $\psi : E \rightarrow R$ assigns types to edges. A graph is homogeneous if ϕ and ψ each map to a single type; otherwise, it is heterogeneous.

Multi-relation graph. A Multi-relation graph is a graph where edges have different types.

Dynamic graph. A dynamic graph is defined as a sequence of graphs G seq = {G1, . . ., GT}, where Gi = (Vi , Ei), for i = 1, . . . , T, where Vi , Ei are the set of nodes and edges for the i-th graph in the sequence respectively.

Temporal Graph Neural Networks for Financial Data

Temporal Graph Neural Networks extend standard GNNs to capture dynamic relationships in time-evolving graphs, making them particularly suitable for financial data (Chen, Ma, & Xiao, 2018). Financial networks exhibit complex temporal dependencies, with transaction patterns varying across different time scales, from intraday fluctuations to seasonal trends. Discrete-time dynamic graph models represent temporal graphs as sequences of static graph snapshots, applying GNN operations to each snapshot independently before integrating temporal information. Continuous-time dynamic graph models directly incorporate time information into the message-passing mechanism, enabling more precise modeling of asynchronous events like financial transactions. Recurrent Graph Neural Networks combine

GNN layers with recurrent architectures like LSTM or GRU to capture sequential dependencies in graph evolution. Attention-based temporal GNNs employ temporal attention mechanisms to identify relevant historical patterns at different time scales. In financial fraud detection, temporal GNNs have been applied to identify suspicious transaction sequences, unusual account behavior patterns, and coordinated fraud rings. These models can detect behavioral changes that indicate account takeover or identity theft by tracking deviations from established temporal patterns. Multi-scale temporal GNNs process information at various time granularities simultaneously, capturing both immediate anomalies and gradual pattern shifts in financial activities. Recent approaches incorporate causal inference techniques to distinguish between genuine behavioral changes and fraudulent activities in temporal financial graphs.

3.2. Fraud Patterns in Graph Structures

Fraudulent behavior manifests as distinctive topological patterns:

Dense Subgraphs: Dense subgraphs offer many meaningful insights in graphs. More often than not, extracting dense sub-graphs (i.e., to find cohesively connected subgraphs with a large density) is of key importance in numerous ap-plication domains. For example, in the co-authorship networks, a dense subgraph could represent a group of researchers with similar research interests. In the product-review bipartite network, extracting dense sub-graphs could help find suspicious fake reviews and detect fraudsters. Moreover, by finding dense subgraphs from a protein-protein interaction (PPI) network, scientists could discover new protein complexes. A major difference behind different dense subgraph detection techniques lies in the specific definition of the density. For example, proposed a greedy algorithm to maximize the average degree (2ms/nS) and extract the densest subgraphs (Chiang, Liu, Si, Li, Bengio, & Hsieh). Moreover, the edge surplus (i.e., the surplus between the number of edges in the extracted subgraph and its corresponding α-quasi-clique) is used to find a denser subgraph than the densest subgraph.

ANOMALY DETECTION: Several literature surveys have been performed on anomaly detection and anomaly detection using graph-based techniques. However, none of these surveys have provided a detailed exploration of graph-based techniques in anomaly detection. In this work, we provide a comprehensive overview of various state-of-art anomaly detection techniques in graph-structured data. We categorize anomaly detection papers based on the type of anomalous component, graph type, method used, and anomaly type.

Camouflage: In camouflage identification, hidden connections between fraudsters are found and redundant connections established from fraudsters to benign nodes are removed. The whole process can be described in three steps. In the first step, part fraudulent neighboring nodes of the fraudster are found through indirect connections. Although there is no direct connection between these fraudulent nodes, CIES-GNN can connect these fraudulent nodes through long-distance meta-paths in higher-order adjacency matrix. In the second step, part fraudulent neighboring nodes of the fraudster are found through potential connections. Taking into account that part fraud nodes still have few connections in higher-order matrix, GAE is used to reconstruct the adjacent matrix of fraud nodes in the higher-order matrix, so potential connections between fraudulent nodes can be found in the generated adjacency matrix (Pourhabibi, Ong, Kam, & Boo, 2020). In the third step, part benign neighbor nodes of

fraudsters are eliminated by removing redundant connections. To avoid fraudsters' feature masked by too many benign nodes when aggregating neighbor nodes, CIES-GNN reconstructs the class-balanced subgraph in a similar way to filter out redundant connections established from fraudsters to benign node.

4. Graph Machine Learning for Fraud Detection

4.1. Graph Neural Networks (GNNs)

In recent years, graphs networks have been used extensively because of their flexibility. Graphs naturally arise in many real-world applications, including social analysis, fraud detection, traffic prediction, computer vision, and many more. Graph based Deep learning models are powerful. Graph structured data are used in graph convolutional models. They are used for prediction, labeling, link prediction, etc. Convolutional Neural Networks (CNN) can process more complex data. We have taken an algorithm which combines both graphs and CNN called Graph Convolutional Networks (GCN) which can work directly on graphs.

We consider a graph G= (N, E) that is generated from the whole bitcoin transaction graph, here *N* represents the transactions, and *E* represents the bitcoin movement. The graph network is constructed from 234,355 edges. First, we extract the graph from its edge list and then we associate its node features for training purpose. Random feature vectors are assigned to generate synthetic features. We use binary labels for classifications. We apply the attention mechanisms by adding more layers to the feature selection. To Create a Feature Matrix, we consider that each row corresponds to a node and each column corresponds to a feature.

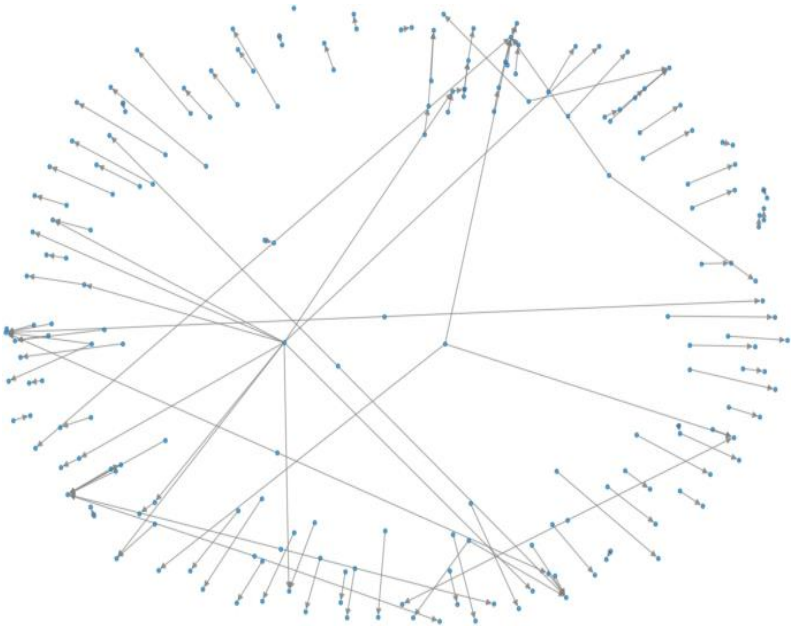

Figure 1: This figure depicts a subgraph corresponding to the Bitcoin data set that is generated from 200 nodes and their transaction. Nodes in the graph usually represent individual transactions.

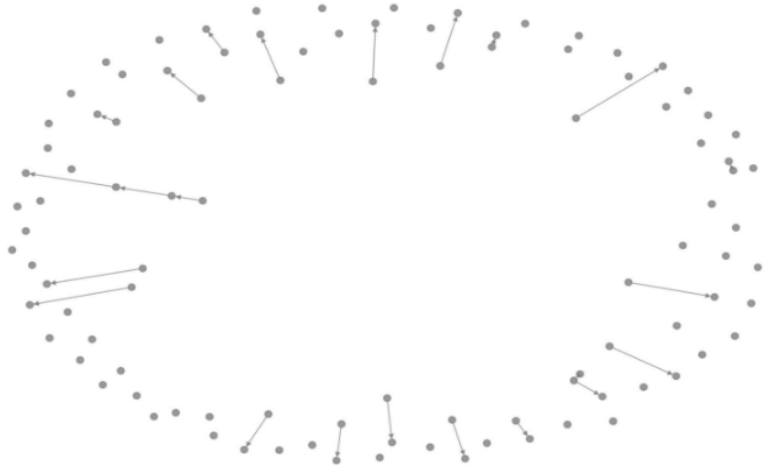

Figure 2: Fruchterman-Reingold layout is another form of graph visulation for better layout. In this image we show the Bitcoin transaction graph for sample of 100 nodes.

Table 2: GNN: Accuracy for different Epoch

From: Graph convolution network for fraud detection in bitcoin transactions

Epoch	Accuracy	Loss	Validation accuracy	Validation loss
1/5	0.1432	3.2758	0.1362	1.7915
2/5	0.82473	0.54122	0.1461	1.8376
3/5	0.94578	0.18972	0.13125	1.8974
4/5	0.98021	0.08273	0.15461	1.92635
5/5	0.9856	0.05921	0.16843	1.93572

We generate the adjacency *A* and the node embedding $H^{(l)}$ matrices corresponding to the graph *G*. We use *A* and $H^{(l)}$ for our GCN at l^{th} layer as input and updates the node using a weight matrix $W^{(l)}$ and gives $H^{(l+1)}$ as output. On a mathematical level. Hence,

$$H^{(l+1)} = \sigma(A' H^{(l)} W^{(l)})).$$

is the activation function for all the layers that we have used, ReLU but the output layer. For the output layer, we have used Softmax as the activation function. The embedding matrix used first is originally from the node features. We use a basic GCN model. A GCN architecture (Figure 3) was proposed by Defferrard et al. and was used by Zonghan et al. In this paper, we use this architecture for our date set. Our GCN has different levels of mechanisms. The node tasks are carried out at end-to-end level. Using semi-supervised learning, we train our GCN model (Zheng, Li, Li, Li, & Gao, 2019). We apply t-Distributed Stochastic Neighbor Embedding (t-SNE) for the reduction in dimensionality and use RelU as an activation function

in our GCN. Our training shows better output for the next level. The values of the accuracy and loss functions for different epochs are shown in Table 2.

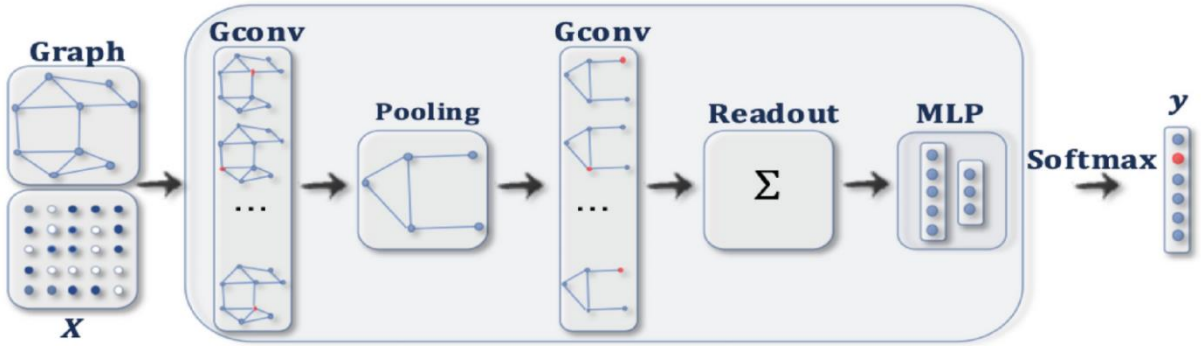

Figure 3: GCN architecture: GCNs are tailored to work with non-Euclidean data, making them suitable for a wide range of applications including social networks, molecular structures, and recommendation systems.

4.2. Evaluation metrics

There are several evaluation metrics for the evaluation of the DL models. In this paper we use Precision, Recall and F1 score for our model and comparison of each of these measures used in different DL models.

Precision= TP/TP+FP

where TP (True Positives) refers to the number of correctly predicted positive instances of skin cancer by a classification model. In this scenario, the classification model is trained to identify whether a given skin lesion or image is indicative of skin cancer (positive class) or not (negative class). When the model correctly predicts a positive instance, it is counted as a True Positive. FP (False Positives) refers to the number of incorrectly predicted positive cases of skin cancer by a classification model (Dal Pozzolo, Caelen, Johnson, & Bontempi, 2015). In this scenario, the classification model is trained to identify whether a given skin lesion or image is indicative of skin cancer (positive class) or not (negative class). When the model correctly predicts a negative instance, it is counted as a False Positive.

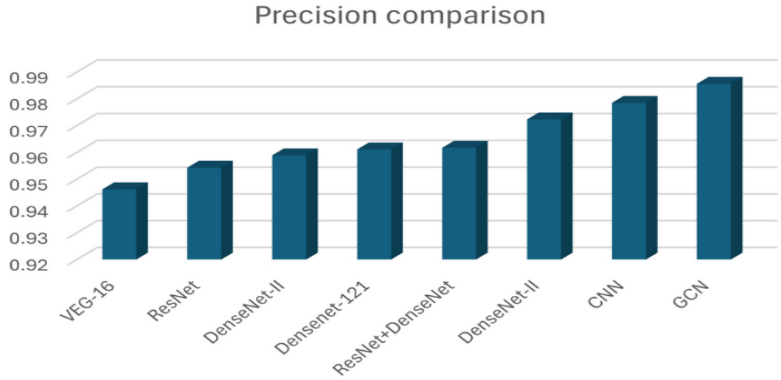

Figure 4: This graph represents the precision values of the comparative models and the proposed GCN model after training for 20 epochs. The findings clearly highlight the superiority of the improved GCN model over the other models, as it achieves the highest accuracy of 98.56%.

$$Recall = \frac{TP}{TP + FN}.$$

The recall measure is depicted, showing a strong correlation with the precision values obtained after training for 20 epochs.

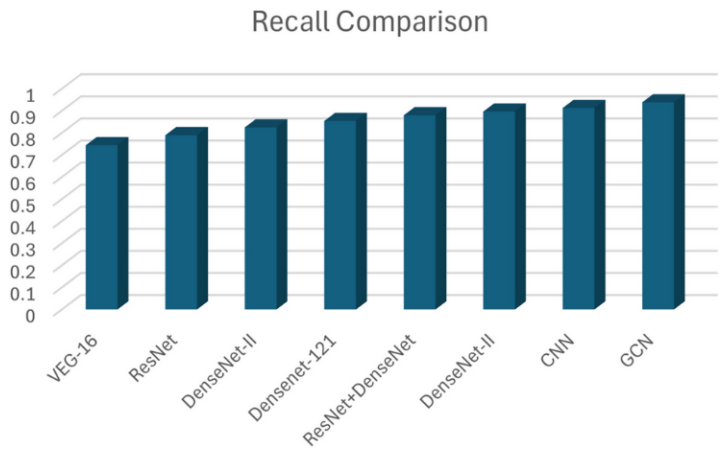

Figure 5: The proposed GCN model stands out with the highest recall of 0.9381, better than other DL models. Recall comparisons of Deep Learning models.

Table 4 Comparison between DL models.

From: Graph convolution network for fraud detection in bitcoin transactions

DL Models	Precision	Recall	F1 score (%)
VEG-16	0.9462	0.7438	75.45
ResNet	0.9543	0.7891	77.83
DenseNet-I	0.9589	0.8245	82.59
DenseNet-121	0.9611	0.8537	87.91
ResNet+DenseNet	0.9617	0.8794	88.63
DenseNet-II	0.9723	0.8962	89.54
CNN	0.9785	0.9126	91.23
GCN	0.9856	0.9381	92.87

$$F1Score = \frac{2 \times Precision \times Recall}{Precision + Recall}.$$

The F1 score, calculated as the harmonic mean of precision and recall, reinforces the trends observed in the previous metric results. Nevertheless, the proposed GCN model shows the highest F1 score of 92. 87% compared to other DL models, which is shown in Figure 6.

From: Graph convolution network for fraud detection in bitcoin transactions

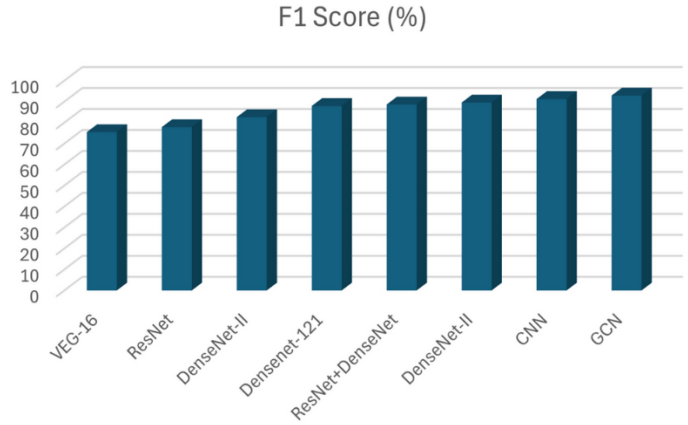

Figure 6: We apply the paired *t* test for CNN and GCN to test their mean difference. The results of the paired *t* test indicated that there is a nonsignificant medium difference between CNN (mean is 31 and standard deviation is 52.1) and GCN (mean is 31.6 and standard deviation is 53.1) with *t* value one and the *p* value is 0.411.

4.3. Graph Attention Networks (GATs)

Graph Attention Networks (GATs) are a type of graph neural network (GNN) that are highly effective for detecting financial fraud. Unlike traditional methods that treat transactions as isolated events, GATs model financial data as an interconnected graph, capturing complex relationships between entities like users, merchants, and accounts. The core innovation of GATs is an attention mechanism that assigns different learned weights to a node's neighbors, allowing the model to focus on the most relevant relationships for fraud detection.

The process of using GATs for financial fraud detection involves several steps:

Graph construction: Financial records are represented as a graph, where nodes can represent entities such as customers, merchants, or transactions. Edges represent the relationships or interactions between them.

Feature representation: Each node and edge is associated with features. For example, a transaction node might have features like the transaction amount, type, location, and time.

Attention mechanism: At each layer of the network, the GAT computes a weighted average of a node's neighbors to create a new, updated representation for that node. The attention mechanism dynamically calculates the weight for each neighbor based on its importance to the central node, rather than treating all neighbors equally like a standard graph convolutional network (GCN) (Defferrard, Bresson, & Van der Gheynst, 2016).

Multi-head attention: To improve stability and performance, GATs use multi-head attention. This involves running the attention mechanism multiple times in parallel and combining the results.

Fraud classification: The final, enriched node representations are then fed into a classifier, which determines whether a given node (e.g., a transaction) is fraudulent.

Advantages of using GATs for fraud detection

GATs offer significant advantages over traditional fraud detection methods:

Captures complex relationships: GATs can uncover sophisticated fraud rings by analyzing multi-hop connections that traditional models would miss. A fraudulent transaction, for instance, might be flagged because of its connection to another suspicious transaction two or three steps away.

Improved accuracy: By focusing on the most relevant connections, GATs achieve higher overall performance metrics, including accuracy, precision, and recall, compared to methods like standard GCNs.

Reduced false positives: The contextual insights provided by GATs help distinguish between legitimate and fraudulent transactions more effectively, which minimizes false alerts.

Enhanced explainability: Certain GAT variants and related tools offer insights into the model's reasoning by highlighting which connections were most important in its decision-making process.

Adaptability: GATs can be used in federated learning frameworks, allowing different financial institutions to train models collaboratively and adapt quickly to new fraud patterns while maintaining data privacy.

Challenges and advancements

Despite their effectiveness, GATs face challenges that researchers are actively working to address:

Over-smoothing: A common problem in deep GNNs, where node representations become too similar after many layers, potentially causing the model to miss subtle fraud indicators.

Advancement: Models like Jump-Attentive GNNs (JA-GNNs) have been developed to create a "jumping mechanism" between layers, preserving crucial information from earlier layers and preventing over-smoothing (Deng, Wang, Zhang, Lian, Chen, & Yu, 2020).

Imbalanced data: Fraud cases are a tiny fraction of all financial transactions. This can bias a model toward classifying everything as non-fraudulent.

Advancement: Techniques such as weighted loss functions or augmented network architectures that incorporate class weighting strategies can be used to address this imbalance.

Temporal dependencies: Fraud patterns evolve over time. Standard GATs may not fully capture these dynamic shifts.

Advancement: Augmented Temporal-aware Graph Attention Networks (ATGATs) and other temporal enhancements are being developed to incorporate time-related features into the model.

High computational time: For very large financial graphs, some GNN-based approaches can require significant computational resources.

Advancement: Researchers are developing more efficient models and leveraging hardware acceleration to reduce execution times, making real-time fraud detection more feasible.

GraphSAGE is used for inductive learning in financial fraud by creating embeddings for nodes (like customers and merchants) based on their features and local graph structure, allowing the model to generalize to new, unseen transactions (Chen, Y.; Wu, L.; Zaki, M., 2020). Its inductive nature is crucial for dynamic financial networks because it can process new data without retraining the entire model, improving scalability and accuracy in tasks like identifying high-risk accounts for fraud detection.

5. Concept Drift Adaptation Mechanisms

Overview of Concept Drift

In this section, we introduce the definition and causes of concept drift, different types of concept drift, and the process of concept drift adaptation methods. Concept drift was first proposed by Schlemmer et al. in 1986 and mainly refers to the fact that the underlying data stream distribution changes over time.

5.1. The Definition of Concept Drift

Assuming that P_{t0} represents the joint probability distribution between the input variable x and the target variable y at time t_0 and P_{t1} represents the joint probability distribution between the x and y at t_1, then concept drift will occur if Equation (1) holds when t_0 turns to t_1.

$$\exists x: Pt0(x,y) \neq Pt1(x,y)$$

At this time, the underlying data distribution no longer conforms to concept C_1, and a new concept C_2 is generated. Due to the characteristics of joint probability $P_t(x, y) = P_t(x)P_t(y|x)$ if Equation (2) is satisfied when t_0 turns to t_1, concept drift will also occur.

$$\exists x: Pt0(x)Pt0(y|x) \neq Pt1(x)Pt1(y|x)$$

Changes in both $P_t(x)$ and $P_t(y|x)$ can lead to concept drift.

5.2. The Causes of Concept Drift

According to the definition of concept drift and the characteristics of joint probability, it can have the following three causes:

Virtual concept drift. When the probability of x changes, but the probability of y under the condition of x does not change, i.e., $P_{t0}(x) \neq P_{t1}(x)$ and $P_{t0}(y|x) = P_{t1}(y|x)$. This case belongs to virtual concept drift, which does not affect its decision boundary and only changes the feature space.

Real concept drift. When the probability of y under the condition of x changes, the probability of x remains the same, i.e., $P_{t0}(y|x) \neq P_{t1}(y|x)$ and $P_{t0}(x) = P_{t1}(x)$. This case has a direct impact on the prediction model and is a real concept drift, which not only changes the feature space but also changes its decision-making boundary.

Hybrid concept drift. In an open environment, both real concept drift and virtual concept drift can exist in the data stream at the same time, i.e., $P_{t0}(x) \neq P_{t1}(x)$, $P_{t0}(y|x) \neq P_{t1}(y|x)$. This is a mixed concept drift, which is most common.

It is worth noting that according to the Bayesian decision theory, we obtain Equation (3):

$$(y|x) = P(y) * P(x|y)P(x)$$

It can be seen that $P_t(y)$ and $P_t(x|y)$ also affect $P_t(y|x)$, thus indirectly causing a real concept drift. The specific manifestations of the concept drift due to different causes are shown in **Figure** 7.

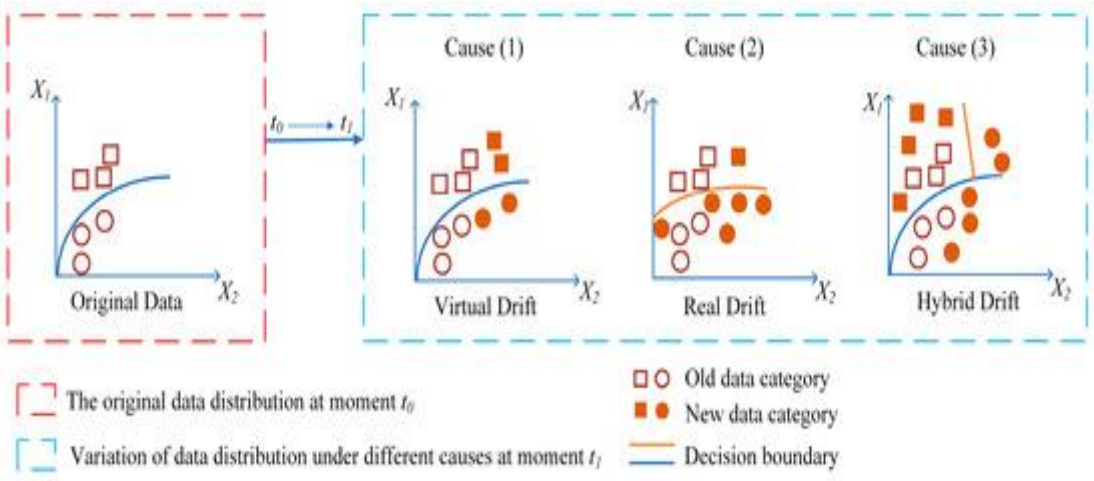

Figure 7. The causes of concept drift in which (X_1, X_2) represents the two-digit feature space and y represents its category label.

For example, in stock trading, users can be divided into profitable and non-profit stocks according to profitability. When a user considers purchasing stocks, a change in the channel of purchase or a small change in the number of purchases does not affect the trend of the stock. However, if affected by an outbreak, the trend of stocks may change, thus directly affecting stock returns. This situation belongs to the real concept drift, so users need to reconsider and make decisions. In real life, virtual drift tends to have less impact on the outcome of a decision. There will be no loss to decision makers. However, real concept drift tends to have a direct impact on decision outcomes due to changes in its data relationships. It requires decision makers to discover in time and re-make decisions to avoid losses.

5.3. The Types of Concept Drift

The changes in concept may manifest in different forms over time. At present, the most popular types of concept drift can be divided into abrupt drift, incremental drift, gradual drift, and recurring drift.

Abrupt drift refers to the rapid change of concept C_1 into concept C_2 in a short period of time, and if an earthquake suddenly occurs in a certain place, its economic model changes instantaneously.

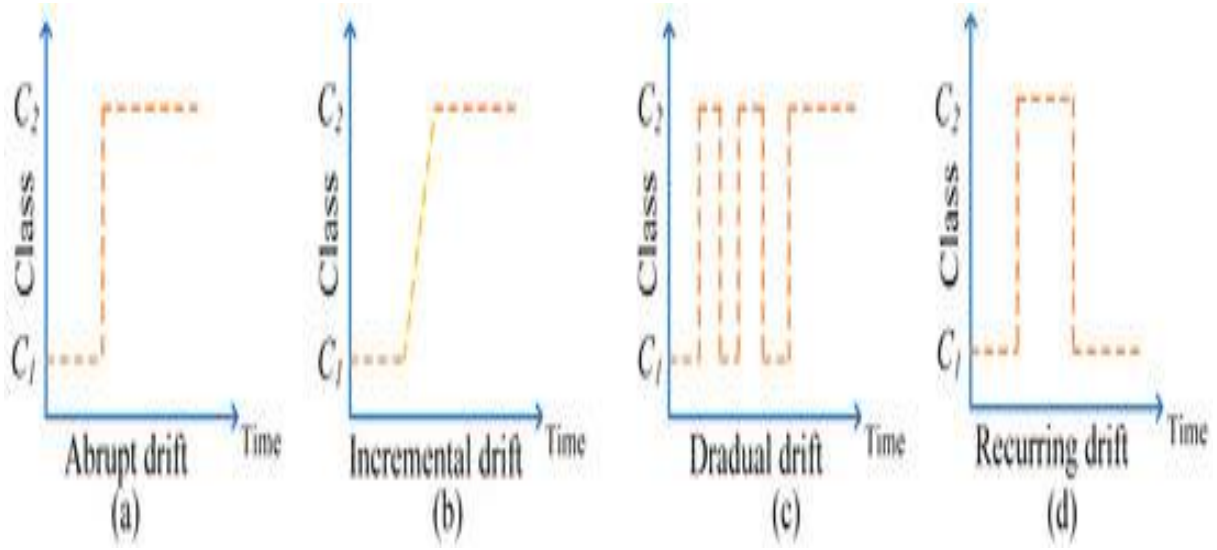

Figure 8. as shown in Figure 8a. Incremental drift refers to the slow transformation of concept C1 into concept C2 in a continuous manner, as the economy gradually recovers after an earthquake, as shown in Figure 8b. Gradual drift refers to a short period of time: C1 and C2 repeatedly switch and eventually stabilize at C2, as the equipment ages, occasionally fails, and finally stops working, as shown in Figure 8c. Recurring drift refers to the fact that over time, the previous concept will reappear after a period of time; for example, the sales of down jackets meet concept C1 in the winter, start to enter the off-season after the end of the winter, their sales will meet concept C2, and then the next winter concept C1 will reappear, as shown in Figure 8d. In addition, the speed of recurring drift can be abrupt, gradual, or incremental. It can also be periodic or irregular.

5.4. The Process of Concept Drift Adaptation Methods under Deep Learning Framework

The general adaptation process of concept drift under the deep learning framework when dealing with unstable state data streams is shown in Figure 9. First, the data stream input (single input or batch input) is generally trained and learned by the deep learning model (single model or ensemble model) to obtain the basic prediction results. Next, if concept drift occurs during this process, a concept drift adaptation method will be triggered to update the deep learning model to accommodate concept drift and maintain the model. The concept drift adaptation method can be divided into two parts: concept drift detection and model update. Among them, concept drift detection contains both active and passive modes, and model updates can be divided into structure updates and parameter updates.

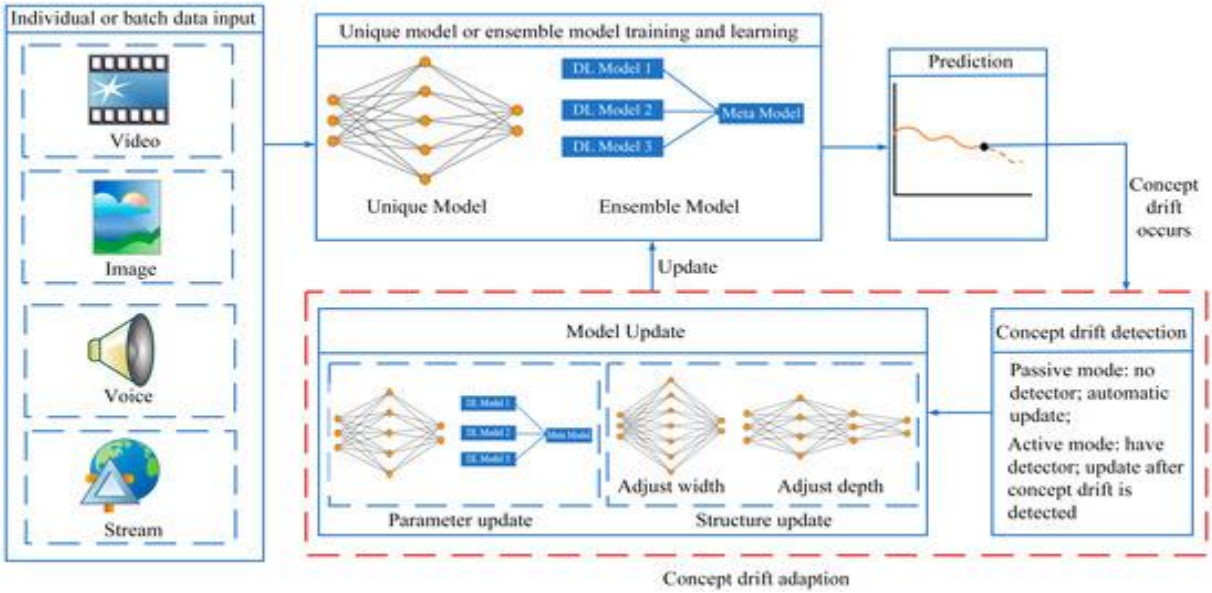

Figure 9. The general process of concept drift adaptation methods under deep learning framework.

Active modes mean that the learning process of a deep learning model contains a concept drift detection algorithm. When concept drift is detected, the concept drift adaptation method will be triggered to update the model. Passive mode means that the method continuously adjusts its model as data are continuously input during the learning process. Instead of using a drift detection algorithm, it uses a concept drift adaptation method to passively update the model continuously. After triggering the concept drift application mechanism, the deep learning model is generally updated to adapt to the concept drift through a model parameter update or a structure update. Model parameter updates can be divided into full parameter updates and partial parameter updates. In particular, parameter updates also include parameter updates between ensemble models. Here, parameter updates are also weight updates. In addition, model structure updates can generally be performed by adjusting the width and depth of the network.

Cross-domain fraud propagation occurs when fraudsters steal credentials or information from one financial service channel or institution to commit fraud on another. This advanced method exploits the multi-channel nature of modern banking and a lack of data-sharing across different domains, making fraudulent activities harder to trace.

6. Unified batch-stream processing with graph updates

The concept of "Unified batch-stream processing with graph updates End-to-End Architecture Patterns in Integration: Adaptive Pipelines + Graph ML in Financial Fraud Detection" describes a sophisticated data architecture for real-time and historical analysis in financial crime prevention (Agrawal, 2024).

6.1. Core Components:

6.1.1. Unified Batch-Stream Processing:

This involves using a single processing engine (e.g., Apache Flink, Apache Spark's Structured Streaming) to handle both continuous, real-time data streams (streaming) and large volumes of historical data (batch) within a unified framework (Lundberg & Lee, 2017). This

eliminates the need for separate architectures, simplifying development and ensuring data consistency.

6.1.2. Graph Updates:

The architecture incorporates dynamic graph databases or graph-processing engines that can be continuously updated with new information from both batch and streaming sources. These updates reflect evolving relationships between entities (e.g., accounts, transactions, individuals) in the financial network.

6.1.3. Adaptive Pipelines:

Data pipelines are designed to be flexible and responsive, adapting to changing data volumes, patterns, and evolving fraud detection models. This might involve dynamic resource allocation, auto-scaling, and the ability to easily integrate new data sources or processing steps.

6.1.4. Graph Machine Learning (Graph ML):

This is a key element for fraud detection. Graph ML models are trained on the dynamic graph data to identify suspicious patterns, anomalies, and hidden connections indicative of fraudulent activities (e.g., money laundering cycles, synthetic identities, coordinated attacks) (Alarab, Prakoonwit, & Nacer, 2020).

6.1.5. End-to-End Integration in Financial Fraud Detection:

The combination of these elements creates an integrated system for financial fraud detection:

6.2. Real-time Ingestion and Graph Updates:

Transaction data, customer information, and other relevant data are ingested in real-time, continuously updating the financial graph.

6.2.1. Feature Engineering with Graph Context:

Graph-based features (e.g., node centrality, community detection, subgraph patterns) are extracted from the evolving graph to enrich the data for ML models.

6.2.2. Graph ML for Fraud Detection:

GNNs or other graph-aware ML models analyze the enriched data to detect fraudulent activities, both in real-time streams and historical datasets (Masihullah, Negi, Matthew, & Sathyanarayana, 2022).

6.2.3. Adaptive Model Deployment and Monitoring:

The system allows for rapid deployment of new or updated fraud detection models, and continuous monitoring of model performance to ensure effectiveness and adapt to new fraud tactics.

6.2.4. Feedback Loops:

Detection results can be fed back into the system to refine the graph structure, update features, and retrain models, creating a continuous improvement cycle.

This architecture enables financial institutions to move beyond rule-based systems to a more proactive, intelligent approach to fraud detection, leveraging the power of graph analysis and machine learning in a unified, adaptive framework.

6.3. Dynamic Graph Construction

6.3.1. Temporal Graph Networks

To review more theoretical information about what is presented in this section, see. A static graph G is defined as the tuple $(V,,\mathcal{V},\mathcal{E})$, where V is the set of vertices, $E \subseteq V \times V$ the set of edges; notice that directed graphs are being considered. Each vertex $v \in V$ has $xv \in \mathcal{V}: \mathcal{V} \in \mathbb{R}d\mathcal{V}$ features and each edge $(u,) \in E$ has $euv \in \mathcal{E}: \mathcal{E} \in \mathbb{R}dE$ features. In addition, $(u) := \{v \in V | (u,v) \in E\}$ is the set of neighborhoods of u in G.

Temporal Graphs: A temporal graph $G\tau$ is defined as the tuple $(V,,V\tau,E\tau)$, where V and E are the set of vertices and edges, respectively, that may appear at any time. Some temporal vertices and edges have time-dependent features such that $V\tau := \{(v,xv,t) | v \in V, xv \in \mathcal{V}, t \in \tau\}$, $E\tau := \{(e,xe,t) | e \in E, xe \in \mathcal{E}, t \in \tau\}$, where $\tau = \{t1,t2,...,tf\}$ is a set of timestamps. The temporal neighborhood of u is given by $\mathcal{N}\tau(u) := \{v \in V | \exists (e,xe,t) \in E\tau, e=(u,v)\}$.

If the timestamps on a temporal graph (TG) follow a fixed time step, the TG becomes a Discrete-Time Temporal Graph.

Discrete-Time Temporal Graph (DTTG): Let $\Delta\tau > 0$ be a fixed time-step and let $t1 < t2 < ... < tn$ be timestamps with $tk+1 = tk + \Delta\tau$. A DTTG GDT is a TG where for each $(v,v,t) \in V\tau$ or $(e,xe,t) \in E\tau$, the timestamps are taken from the set of fixed timestamps.

6.3.2. Dynamic Graph Neural Networks, Discrete and Continuous Case

Temporal graphs can be represented as a stream of static graphs or vertex/edge events. The former is referred to as a Snapshot-based Temporal Graph, while the latter is known as an Event-based Temporal Graph. This work focuses on designing an algorithm based on Event-based Temporal Graphs, i.e., a continuous dynamic approach.

Snapshot-based Temporal Graph (STG): Let $t1 < t2 < ... < tn$ be an ordered set of timestamps in $G\tau$.

Let $Vj := \{(v,xv,ti,tf) | v \in V\tau, ti \leq tj \leq tf\}$, $Ej := \{(e,xe,ti,tf) | e \in E\tau, ti \leq tj \leq tf\}$ be the sets of vertices and edges that exist between an initial time ti and a final time tf and $Gj=(Vl,El)$, $l=1,2,...,n$ be the graph snapshots. A Snapshot-based Temporal Graph is the sequence $GS\tau = \{(Gl,tl) | l=1,2,...,n\}$.

The literature indicates STGs are related to DTTGs because the snapshots are at periodic fixed time intervals.

Another form of graph representation occurs when the data model a continuous process, i.e., a dynamic continuous temporal graph. However, this approach is better modeled as a stream of events, so it receives the name of an Event-Based Temporal Graph.

Event-based Temporal Graph (ETG): Let $G\tau$ be a temporal graph and $\mathcal{E}$ one of the following events:

Vertex insertion $\mathcal{E}+ := (v,t)$: There is an addition of the vertex v to $G\tau$ at time t.

Vertex deletion $\mathcal{E}- := (v,t)$: There is a removal of the vertex v from $G\tau$ at time t.

Edge insertion $\mathcal{E}+ := (e,t)$: There is an addition of the edge e to $G\tau$ at time t.

Edge deletion $\mathcal{E}-:=(e,t)$: There is a removal of the edge e from $G\tau$ at time t.
An ETG of a temporal graph is a sequence of events
$GE\tau=\{\mathcal{E}|\mathcal{E}\in\{\mathcal{E}+V,\mathcal{E}-V,\mathcal{E}+E,\mathcal{E}-E\}\}$.

Message-Passing TGNN—MPTGNN: Let $G\tau$ be a temporal graph with parameters $V\tau$ and $E\tau$. At each iteration of the MPTGNN, an embedding $hkv(t)$ is updated with neighborhood information $\mathcal{N}\tau(v)$. The message passing can be expressed as
$h(k+1)v(t)$=UPDATE$k(h(k)v(t)$,AGG$k(\{h(k)u(t),\forall u\in\mathcal{N}\tau(v)\}))$
=UPDATE$k(h(k)v(t),m(k)\mathcal{N}(u)(t))$, (Pourhabibi, Ong, Kam, & Boo, 2020)

where UPDATE is a differentiable function (a neural network); AGG could be some permutation-invariant operation such as mean, max-pooling, sum, etc.; $m\mathcal{N}(u)(t)$ is the message that aggregates temporal neighborhood information about the vertex *n* and $\mathbf{h}(0)u(t)=su(t)$ is the state of *v* at time *t*. The TGNN algorithm proposed uses the concept of memory as a set of vectors that summarizes the history of a vertex and is updated whenever an event occurs. Given a set of events $\mathcal{E}$, the state of *v* is updated based on $\mathcal{E}$ as
$\mathcal{M}v(t)$=MEMAGG$(\{(su(t),sv(t),t-tv,\mathcal{E}uv(t))|(u,v,t)\in\mathcal{E}\})$
$s(t+)$=MEMUPDATE$(sv(t),\mathcal{M}v(t))$,

where $sv(0)=xv$. According to the previous definitions, the embeddings in a TGNN can be expressed as
$h(k+1)v(t)$=UPDATE$k(h(k)v(t)$,AGG$\{\mathcal{M}\mathcal{E},\mathcal{E}=\mathcal{E}\pm E=(u,t')$with$u\in\mathcal{N}\tau(v)\})$,
where $\mathcal{E}\pm E=(u,t')$ with $u\in\mathcal{N}t(v)$ is the addition or deletion of a temporal neighbor of *v*. One thing to clarify is that the MPTGNN belongs to the TGN approach; from now on, this terminology will be used (Ramanathan, V.; et al., 2022).

6.4. Real-time Graph Embedding Updates

Real-time graph embedding updates enhance financial fraud detection by continuously learning new patterns from dynamic transaction data streams. This approach treats financial networks as continuously evolving graphs, where new nodes and edges (transactions) are constantly added (PayPal Engineering, 2021). By dynamically updating embeddings—low-dimensional vector representations of graph entities—the system can detect fraudulent activities that traditional, static models would miss.

6.4.1. How real-time graph embedding works for fraud detection

The process moves beyond batch processing, in which models are retrained periodically, to a continuous learning cycle.

Graph construction: A dynamic, heterogeneous graph is built from financial data, representing various entities as nodes and their interactions as edges. For example:

Nodes: Customers, merchants, IP addresses, and devices.

Edges: Transactions, login attempts, or other interactions.

Streaming data ingestion: New transactions and user activities arrive as a continuous data stream.

Real-time embedding updates: When a new event occurs, the graph embeddings are updated incrementally. Several techniques achieve this:

Temporal Graph Networks (TGNs): These models use a memory module to store the evolving state of nodes. When a new interaction occurs, the memory for the involved nodes is updated, influencing their embeddings.

Streaming Graph Neural Networks (SGNNs): This approach updates node features based on new interactions, considering the order of events and the time between them. It then propagates this new information to affected nodes.

Lambda Neural Networks (LNNs): Part of the BRIGHT framework, these networks decouple the inference process into batch and real-time stages to minimize computational overhead.

Fraud detection: The system uses the continuously updated embeddings to identify suspicious behavior in real time.

Anomaly detection: Transactions that cause a sudden, significant shift in a node's embedding can be flagged as anomalous.

Network pattern detection: The updated embeddings help identify complex fraud rings that operate with money mules or fake IDs, which are often missed by traditional methods.

Classifier integration: Updated graph embeddings are fed into a classification model to produce a risk score for each new transaction (Ribeiro, Singh, & Guestrin, 2016).

6.5. Advantages over traditional methods

Adapts to new fraud patterns: The model incorporates new fraud signatures as they emerge, without needing a full, time-consuming retraining cycle.

Captures higher-order relationships: By analyzing the entire network, GNNs can detect fraud based on multi-hop connections, not just individual transaction features. This allows detection of "sleeper" accounts that appear benign until they are linked to a broader fraud network.

Reduces false positives: Analyzing transactional context and network structure helps the system to distinguish between normal and abnormal behavior more accurately.

Enhanced scalability: Modern GNN architectures are built to handle massive, high-velocity data streams more efficiently than static models.

Balances temporal and structural data: Solutions like DyHDGE are specifically designed to balance the importance of both the temporal order of events and the spatial structure of the graph.

Implementation examples and technologies

AI Blueprints (NVIDIA): A developer-focused resource that provides a solution for building GNN-based real-time fraud detection systems. It combines GNNs with other models like XGBoost for high accuracy and scalability.

FinGraphFL: A federated learning approach that uses Graph Attention Networks (GATs) to detect fraud across multiple financial institutions without sharing sensitive data. It continuously refines the model while protecting privacy (Wu, He, Gao, & He, 2021).

BRIGHT: A framework that uses a two-stage directed graph to enable efficient real-time inference by limiting message-passing to historical data.

DyHDGE: A network designed to extract both temporal and structural information from dynamic heterogeneous transaction graphs.

Graph databases: Services like Amazon Neptune and Neo4j provide the infrastructure for storing and querying complex graph data in real time.

7. Challenges and Open Problems

7.1. Current Challenges and Limitations

Despite its importance, real-time fraud detection faces several challenges and limitations. One of the primary issues is the balance between detecting fraud effectively and minimizing false positives, which can lead to legitimate transactions being incorrectly flagged and declined. Additionally, the sophistication of fraudsters, who continually evolve their tactics to evade detection, poses a significant challenge. The complexity and volume of transactional data, coupled with the need for real-time analysis, also require substantial computational resources and advanced technological infrastructure.

The fight against fraud is constantly evolving. Experian's recent fraud research shows that 71% of the fraud leaders in our survey are struggling to keep up with the rapidly evolving fraud threat. But as new technology creates opportunities for fraudsters, so too does it enable better fraud prevention.

This balance between fraud attacks and prevention is highly dynamic, and the only way to stay ahead of fraudsters is to take advantage of the latest fraud detection technology. The demand for new technology is clear, with 73% of businesses in the survey seeing their fraud losses increase in the past 12 months.

Our research indicates that Artificial Intelligence (AI) and Machine Learning (ML) are now pivotal technologies in this fight, and when combined with device data and biometrics, can help swing the balance in businesses' favour. In this article, we take a detailed look at the top five challenges that are limiting fraud prevention and explore which fraud solutions can fill these gaps.

8. Future Research and Practical Applications

This review highlights several important directions for future research while also offering practical implications for auditors, regulators, and firms. From a research perspective, longitudinal studies are needed to evaluate the sustained effectiveness of anti-fraud systems over time and across industries with different risk profiles. Equally important is advancing the interpretability and transparency of sophisticated AI models to ensure that they can be trusted and adopted by auditors and regulators. A key gap in the literature is the absence of standardized frameworks that allow stakeholders to systematically evaluate fraud risks. While machine learning models achieve impressive predictive performance, their "black box" nature raises concerns about accountability and regulatory acceptance. Future research should therefore focus on developing explainable, user-friendly tools that can be integrated into auditing practice without requiring deep technical expertise.

A promising avenue is the design of fraud risk scoring systems that combine traditional financial ratios with analytics-based indicators. Such systems could generate composite fraud risk scores—expressed in intuitive percentage terms—that auditors and internal control

professionals can readily apply in decision-making. This approach would bridge the gap between conventional ratio analysis and advanced machine learning, producing tools that are both empirically robust and operationally accessible.

On a practical level, several concrete recommendations emerge. Auditors should integrate advanced analytics into their standard procedures, supplementing traditional sampling and ratio analysis with anomaly detection and machine learning models, while also investing in training to interpret outputs responsibly. Regulators should establish guidelines for the ethical use of AI in fraud detection, with particular attention to issues of algorithmic bias, model transparency, and auditability. They can also promote the industry-wide adoption of standardized fraud risk scoring frameworks to ensure comparability and fairness. Firms should prioritize investments in high-quality data infrastructures, hire or upskill staff with cross-disciplinary expertise in accounting and data science, and foster a culture that supports data-driven decision-making. Collaborative efforts between firms, auditors, and regulators are essential to ensure that fraud analytics technologies are implemented responsibly, effectively, and in ways that strengthen the integrity of financial reporting.

9. Conclusions

In today's digital era, businesses should adapt to changes quickly to be competitive in the market. The streaming data pipeline empowers the organization to process and analyze the data as soon as it is ingested to the streams. Streaming Data Pipeline helps organizations to prevent fraud, optimize IoT operations, real-time analysis, and many more.

As technology advances, streaming data pipelines are becoming essential for organizations to build a safe and reliable system that prevents their users from frauds as well as generating real-time insights to grow their business.

Real-time data pipelines are now a strategic necessity, not an optional enhancement. They deliver the speed, responsiveness, and intelligence modern organizations need to stay competitive. With advances in cloud streaming, AI, declarative pipelines, and federated data governance, businesses can unlock transformative operational gains with robust, low-latency systems.

Research on the detection of financial fraud by applying ML techniques is a significant topic. On the one hand, fraud directly affects the business world and, on the other hand, detecting it early involves great challenges; this has led to designing tools using AI, such as ML techniques. This study evaluates the efficacy of GNNs in the detection of financial fraud by addressing a set of comprehensive research questions that encompass the methodologies, roles, design, deployment, and challenges associated with GNNs within the financial fraud detection field. The introduction of a unified framework helps to systematize the diverse approaches observed in existing literature, offering a clearer understanding of how GNNs can be effectively applied in complex financial contexts. Our findings contribute to the body of knowledge by detailing the adaptive capabilities of GNNs to the dynamic nature of financial networks and emphasizing their potential to enhance fraud detection mechanisms. Moreover, this work identifies existing gaps and outlines prospective research directions that could further solidify the role of GNNs in improving fraud prevention systems (Fiore, De Santis, Perla, Zanetti, &

Palmieri, 2019). The insights provided here aim to guide future studies toward more nuanced applications and integration strategies for GNNs in real-world financial systems.

References:

Agrawal, V. (2024, December 20). *Understanding Streaming Data Pipelines*. Retrieved from Learn | Hevo; Hevo Data: https://hevodata.com/learn/streaming-data-pipelines/

Alarab, I., Prakoonwit, S., & Nacer, M. I. (2020). Competence of graph convolutional networks for anti-money laundering in Bitcoin blockchain. *Proceedings of the 5th International Conference on Information System and Data Mining*, 23–27.

Chen, J., Ma, T., & Xiao, C. (2018). *FastGCN: Fast learning with graph convolutional networks via importance sampling*. Retrieved from International Conference on Learning Representations.

Chen, Y.; Wu, L.; Zaki, M. (2020). Iterative deep graph learning for graph neural networks: Better and robust node embeddings. *Advances in Neural Information Processing Systems*, 19314–19326.

Chiang, W. L., Liu, X., Si, S., Li, Y., Bengio, S., & Hsieh, C. J. (n.d.). (2019). Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks.

Dal Pozzolo, A., Caelen, O., Johnson, R. A., & Bontempi, G. (2015). Calibrating probability with undersampling for unbalanced classification. *2015 IEEE Symposium Series on Computational Intelligence*, 159–166.

Defferrard, M., Bresson, X., & Van der Gheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in Neural Information Processing Systems*, 3844–3852.

Deng, S., Wang, X., Zhang, H., Lian, D., Chen, E., & Yu, Z. (2020). Adversarial attacks and defenses for graph neural networks: Challenges, methods, and opportunities. *arXiv preprint arXiv*, 2009.12119.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *International Conference on Machine Learning*, 1126–1135.

Fiore, U., De Santis, A., Perla, F., Zanetti, P., & Palmieri, F. (2019). Using generative adversarial networks for improving classification effectiveness in credit card fraud detection. *Information Sciences*, 479, 448–455.

IEEE Computational Intelligence Society. (2019). IEEE-CIS fraud detection dataset. *Kaggle Competition*, https://www.kaggle.com/c/ieee-fraud-detection.

Kumar, S., Hooi, B., Makhija, D., Kumar, M., Faloutsos, C., & Subrahmanian, V. (2018). Rev2: Fraudulent user prediction in rating platforms. *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 333–341.

Liu, Y., Ao, X., Qin, Z., Chi, J., Feng, J., Yang, H., et al. (2021). Pick and choose: A GNN-based imbalanced learning approach for fraud detection. *Proceedings of the Web Conference 2021*, 3168–3177.

Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 4765–4774.

Masihullah, S., Negi, M., Matthew, J., & Sathyanarayana, J. (2022). Identifying fraud rings using domain aware weighted community detection. *Machine Learning and Knowledge Extraction*, 108–122.

PayPal Engineering. (2021). Graph neural networks for fraud detection at PayPal. *PayPal Engineering Blog*, https://medium.com/paypal-tech.

Pourhabibi, T., Ong, K. L., Kam, B. H., & Boo, Y. L. (2020). Fraud detection: A systematic literature review of graph-based anomaly detection approaches. *Decision Support Systems*, 133, 113303.

Ramanathan, V.; et al. (2022). Scaling graph neural networks for fraud detection in production. *KDD Workshop on Mining and Learning from Time Series*.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?" Explaining the predictions of any classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1135–1144.

Wu, Y., He, H., Gao, D., & He, X. (2021). Federated learning for privacy-preserving fraud detection. *IEEE International Conference on Big Data*, 2133–2142.

Zheng, L., Li, Z., Li, J., Li, Z., & Gao, J. (2019). AddGraph: Anomaly detection in dynamic graph using attention-based temporal GCN. *International Joint Conference on Artificial Intelligence*, 4419–4425.