# Anomaly Detection in Large-Scale Evolving Networks using Temporal GNNs

Luke Cashion
McGill University
Canada
luke.cashion@mail.mcgill.ca

Luca Louis
McGill University
Canada
luca.louis@mail.mcgill.ca

## ABSTRACT

We present a scalable framework for anomaly detection in large-scale evolving networks using temporal Graph Neural Networks (GNNs). Our approach combines a prediction-based encoder-decoder GNN that forecasts network structure with an online embedding similarity detector for real-time anomaly scoring. The methods are evaluated on the Elliptic Bitcoin transaction network, a dynamic graph with labeled illicit nodes. This final report extends our progress work by more specifically detailing the methodology, final results, discussion and conclusion sections. Preliminary experiments with a static node embedding baseline yield F1 scores around 0.65, confirming the challenge of the task. We anticipate that our temporal GNN will improve detection accuracy (e.g., higher ROC-AUC) by capturing temporal patterns that static methods miss. We outline the experimental setup, discuss evaluation metrics and threshold tuning, and propose future directions such as adaptive sampling, unsupervised learning, and broader dataset generalization to further enhance the system.

**ACM Reference Format:**
Luke Cashion and Luca Louis. 2025. Anomaly Detection in Large-Scale Evolving Networks using Temporal GNNs. In . ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION AND MOTIVATION

Modern systems generate vast interaction networks—financial transaction ledgers, communication networks, social media graphs—that evolve over time. These dynamic networks can harbor fraudulent transactions, cyber-attacks, or misinformation campaigns which must be identified promptly. The motivation for our work is to develop techniques that monitor large, continually updating graphs and detect anomalies in real-time.

Traditional data analysis assumes independent samples or a static dataset, whereas network data is *relational* and *dynamic*. A graph's behavior is influenced by its connectivity and how that connectivity changes. Static analysis may miss temporal patterns that span multiple snapshots. Detecting dynamic structural changes—especially sudden or evolving ones—requires approaches combining graph topology with temporal evolution.


*Conference'17, July 2017, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Our work addresses three key challenges in temporal network anomaly detection:

- **Real-time Processing**: Networks evolve continuously, requiring algorithms that can detect anomalies on streaming data without storing full history. This necessitates efficient, incremental updates to network representations.
- **Structural Evolution**: Anomalies may manifest as unusual patterns in how network structure changes over time, such as sudden bursts of new connections or unexpected community formations. These patterns often span multiple time steps and require temporal context to identify.
- **Imperfect datasets**: Real world datasets for dynamically evolving graphs for anomaly detection are frequently heavily imbalanced and have scarce labelling. Most datasets have a large majority of nodes unlabelled (illicit vs licit) forcing having to balance both supervised and unsupervised approaches.

We propose a dual-pipeline approach that combines:
(1) *A prediction-based temporal GNN*, which learns to forecast the network's evolution and flag deviations. This model captures structural patterns via graph convolution and temporal dependencies via recurrent or attention mechanisms.
(2) *An online similarity-based detector*, which keeps track of each node's evolving embedding and raises an alert whenever there is an abrupt change relative to past behaviours. We use cosine similarity between nodes to track behavioural changes.

This hybrid approach is designed to capture both *structural anomalies* - by predicting the next-step topology via the GNN - and behavioral anomalies - by continuously tracking node or edge-level deviations through an online embedding monitor.

Our key contributions in this work include:

- A novel architecture combining a forecasting temporal GNN with a similarity-based online detector for anomaly detection in evolving graphs. We compute an anomaly score based on both of these (reconstruction error + embedding change)
- An efficient online scoring mechanism that maintains interpretable edge-level anomaly scores in real time.
- A comprehensive evaluation plan and framework for temporal network anomaly detection, including comparisons to static and streaming baselines on a large financial transaction graph.

The remainder of this paper is organized as follows. Section 2 reviews related work on graph anomaly detection, with emphasis on temporal GNN approaches. Section 3 formalizes the problem. Sections 4 and 5 describe the dataset and our methodology (baseline methods and the temporal GNN framework). Section 6 outlines the experiment setup. In Section 7, we present preliminary results, and

in Section 8 we discuss the evaluation metrics, threshold tuning, and the expected advantages of the temporal GNN over baselines. Finally, Sections 9 and ?? conclude the paper and suggest future research directions.

## 2 RELATED WORK

### 2.1 Surveys and Foundational Approaches

Graph anomaly detection has been surveyed extensively in recent years. Qiao and Tong [5] provide a taxonomy of techniques including structural, attribute, and temporal methods, highlighting the rise of unsupervised and deep learning approaches. They note that many early methods (e.g., community detection or statistical outlier models) struggle to handle evolving graph data. A comprehensive 2024 survey by Tian et al. [9] also covers dynamic graph anomaly detection, underscoring the need for models that capture temporal dependencies and scale to large, streaming graphs. Our work builds on these surveys by focusing on real-time and predictive modeling aspects that have been less explored.

### 2.2 Temporal GNNs for Dynamic Graph Anomaly Detection

The advent of temporal GNNs has led to significant advances in anomaly detection on evolving networks. Kim et al. [4] developed a Temporal Graph Network (TGN) with memory modules to handle dynamic edge updates in financial transaction graphs, achieving improved fraud detection AUC compared to static GNN baselines.

Recent approaches leverage attention and transformer mechanisms: Huang et al. [7] introduced the HO-GAT model to simultaneously detect anomalous nodes and motifs, while Liu et al. [8] proposed TADDY, a transformer-based model for detecting anomalous edges across snapshots.

Label scarcity in dynamic graphs remains a core challenge. SAD (Semi-supervised Anomaly Detection), introduced by Tian et al. [9], utilizes unlabeled data distributions during training to detect novel anomalies. Similarly, Cavallo et al. [10] present a multi-task temporal GNN for node and edge anomaly detection, combining memory modules and attention layers to achieve near-0.9 AUC in intrusion detection tasks.

### 2.3 Anomalies in Temporal Knowledge Graphs

In event-based and semantic graphs, Reha et al. [6] proposed an anomaly detection method that handles event streams in continuous time. Their work highlights the importance of rich semantics and fine-grained timestamps. While our work focuses on homogeneous financial transaction graphs, future extensions may adopt insights from these domains to handle typed nodes and edges in evolving contexts.

### 2.4 Other Approaches for Evolving Graph Anomalies

Some scalable methods for anomaly detection in graph streams rely on sketching and heuristic scoring. MIDAS [11] uses count-min sketches to detect sudden bursts in edge activity with theoretical guarantees and real-time performance. These approaches are non-learning based but offer strong performance on high-speed data.

In contrast, generative models and autoencoders reconstruct graphs and use reconstruction loss as an anomaly signal. Traditional anomaly detection methods (e.g., LOF, Isolation Forest, One-Class SVM) also remain widely used as baselines. Our system benchmarks against such baselines with streaming and learned GNN-based detectors.

### 2.5 Our Research

Our Implemented method largely builds on known ideas in dynamic graph anomaly detection, but puts them together in a unique way. a GNN encoder/decoder with a GRU-based temporal memory for link prediction – is very much in line with prior work. For example, Kim et al.'s Temporal Graph Network (TGN) uses memory modules to update node embeddings over time. Likewise, methods like AddGraph or Cavallo et al.'s approach employ a GNN (often a GCN or GAT) per snapshot combined with a recurrent unit to capture temporal evolution, then use that to predict edges or score anomalies.

Where our framework does differentiate itself is in the hybrid anomaly scoring. we explicitly combine a link prediction model with an embedding shift detector which is not approach we found anywhere else. For instance, HO-GAT (Huang et al.) is an attention-based autoencoder that finds anomalous nodes/subgraphs in a static or snapshot setting, and TADDY (Liu et al.) uses a transformer over snapshots with a clustering-based anomaly detection for edges. Those methods don't include a real-time embedding drift monitor. This could be meaningful is our addition enables us to catch anomalies that pure link prediction might miss.

## 3 PROBLEM DEFINITION

We formalize the problem as detecting anomalous nodes or edges in an evolving graph. Let $\{G^{(t)}\}_{t=1}^{T}$ be a sequence of graph snapshots over discrete time steps $t = 1, 2, \ldots, T$, where each snapshot $G^{(t)} = (V^{(t)}, E^{(t)})$ consists of a set of nodes $V^{(t)}$ and edges $E^{(t)}$. We assume that $V^{(t)}$ may include new or removed nodes over time, and similarly for edges in $E^{(t)}$.

For a given node $v$ (or edge $(u, v)$) at time $t$, let $\text{score}(v^{(t)})$ be an anomaly score that quantifies how unexpected the node's behavior is at $t$ relative to historical patterns (similarly $\text{score}((u, v)^{(t)})$ for an edge). We flag an anomaly if the score exceeds a threshold $\tau$:

$$\text{score}(v^{(t)}) > \tau \quad \text{or} \quad \text{score}((u, v)^{(t)}) > \tau. \tag{1}$$

The threshold $\tau$ can be learned or set based on statistical properties of score distributions (e.g., a high percentile of past scores).

Our framework considers two types of anomalies:

- **Node-level anomalies**: e.g., a normally low-activity node suddenly conducts an unusually large number of transactions or forms an unexpected pattern of connections.
- **Edge-level anomalies**: e.g., an unexpected connection between two nodes that do not typically interact, or an edge whose attributes deviate significantly from normal patterns.

*Anomaly Score Computation.* Our approach calculates two complementary signals at each time step, then fuses them into a single score for each node (or edge). First, a *reconstruction error* highlights edges that actually occur but were assigned low probability by the

decoder, reflecting structural deviations from expected connections. Second, an *embedding shift* tracks how far a node's current embedding strays from its own historical embedding; a sudden drop in similarity implies a behavioral change. By taking a weighted sum of these signals, we produce a final anomaly score. Whenever this combined score for a node or edge exceeds the threshold $\tau$, it is flagged as anomalous, capturing both unexpected network structure and abrupt embedding changes in one unified framework.

# 4 DATASET DESCRIPTION

We use the **Elliptic Bitcoin Dataset** [1] as the primary testbed for our study. This dataset consists of a large transaction network from the Bitcoin blockchain, containing 203,769 nodes and 234,355 directed edges (transactions) over $T = 49$ discrete time steps. Each node represents a transaction, and edges represent flows of cryptocurrency from one transaction to another. Node features are provided as a 166-dimensional vector capturing inherent properties (e.g., time step, aggregated transaction amounts, local network metrics, etc.). Importantly, a subset of nodes are labeled as licit (legitimate) or illicit (malicious, e.g., money laundering or fraud), while the majority are unlabeled (unknown status).

The temporal and partially labeled nature of Elliptic makes it well-suited for anomaly detection in a financial context. Illicit nodes can serve as examples of anomalies, though not all anomalies in a graph need be illicit (and not all illicit nodes are easily detectable without temporal analysis). A key challenge is that about 75% of the nodes are of unknown label, which means our anomaly detection must largely rely on unsupervised signals.

The full graph is large and evolving, which poses computational challenges for iterative model training. To manage this, we apply a subgraph sampling strategy during development: for each time step, we sample a small fraction (e.g., 0.1%) of nodes and their associated edges to include in the training or evaluation batch. This retains a mix of licit and illicit instances across the timeline while keeping computations tractable. This random subsampling of the graph at each step allows rapid prototyping of models.[1] We ensured that sampled subgraphs still preserve meaningful structural and temporal diversity. In future work, we plan to explore more advanced sampling techniques (e.g., importance sampling focusing on high-degree or recently active nodes) to better retain important patterns while still controlling scale.

For our experiments, we ignore edges and nodes beyond a certain time horizon when training the temporal models to simulate an online scenario. We also handle the class imbalance (illicit vs licit) by appropriate metrics and threshold tuning (described in Section 8). Unknown-label nodes are generally not used in computing supervised loss or final evaluation metrics, but they are included in the graph structure so that their presence influences the unsupervised anomaly scoring of connected transactions.

# 5 METHODOLOGY

Our anomaly detection system is implemented using PyTorch Geometric (PyG) and Deep Graph Library (DGL) for flexibility in handling graph data. We have built a modular pipeline comprising data preprocessing, baseline anomaly detection methods, the temporal GNN model, and evaluation utilities. In this section, we describe the baseline methods used for comparison and then detail our temporal GNN framework.

[1]See [? ] for details.

## 5.1 Data Ingestion and Preprocessing

We preprocess the Elliptic dataset by first segregating the transaction graph into temporal snapshots. Each discrete time step $t$ (1 through 49) corresponds to a subgraph $G^{(t)}$. We construct a dictionary of graphs indexed by time: `graphs_by_t = {1:` $G^{(1)}$`, 2:` $G^{(2)}$`, ..., 49:` $G^{(49)}$`}`, where each $G^{(t)}$ is represented in PyG/DGL format (with node feature matrix and edge list). Node features are attached from the dataset's CSV files, and edges are added according to the transaction list for that time. We perform consistency checks to handle any missing data or timestamp irregularities (ensuring, for example, that no transaction is assigned to a time step outside the range 1–49). Features are normalized when appropriate (e.g., log-scaling transaction amounts) to stabilize training.

Because of the dataset's size, we apply the subsampling strategy mentioned: at load time, each $G^{(t)}$ can be subsampled to include a fraction of nodes. We typically maintain the chronological order of transactions and do not mix data from future time steps into the past. This preprocessing yields a sequence of (potentially sampled) graphs ready for input into baseline models or the temporal GNN.

## 5.2 Baseline Implementations

To evaluate the effectiveness of our temporal GNN, we compare against two baseline anomaly detection strategies: a static graph embedding approach, and simple streaming heuristics. These baselines provide reference points for performance if one were to ignore temporal information or use only rudimentary temporal logic.

*5.2.1 Static Embeddings + Outlier Detection.* In the static baseline, we condense the graph (or a portion of it) into a single snapshot and apply standard graph embedding and outlier detection techniques, essentially ignoring temporal order. For example, we take the graph at a chosen time step (or an aggregate of several early time steps) as $G_{\text{static}}$ and compute node embeddings for $G_{\text{static}}$. We experiment with two embedding methods:

- **Node2Vec**: a random-walk-based embedding method that generates a low-dimensional representation for each node based on network neighborhoods.
- **GraphSAGE (static)**: a GraphSAGE [3] model without temporal updates, which can inductively generate embeddings for nodes using their one-hop (and multi-hop) neighborhood features. GraphSAGE is appealing for large graphs since it can generalize to unseen nodes and has been shown to scale to inductive scenarios.

Once we obtain node embeddings (e.g., 128-dimensional vectors), we apply outlier detection algorithms in the embedding space to score each node. We implemented and tuned the following:

- **Isolation Forest**: an ensemble anomaly detector that isolates points by randomly partitioning feature space. It produces an anomaly score based on the average path length required to isolate a point in a tree ensemble.
- **Local Outlier Factor (LOF)**: a density-based detector that assigns each node a score (LOF) representing how isolated it

is with respect to the density of its neighbors. A high LOF indicates the node is in a sparse region (potential outlier).
- **One-Class SVM**: we also experiment with a one-class Support Vector Machine that attempts to learn the boundary of "normal" nodes in embedding space and classify points outside that boundary as anomalies.

These methods output a real-valued anomaly score for each node in $G_{\text{static}}$. We then apply a decision threshold to label anomalies (as per Equation 1). Using the partial ground-truth labels (licit/illicit) available in the Elliptic data, we can evaluate the precision, recall, F1-score, and ROC-AUC of these detections.

We optimized this baseline by tuning key hyperparameters. For Node2Vec, we tried walk lengths of 20–30, number of walks per node 10–20, and varying the return/exit parameters $p$ and $q$ (to balance breadth-first vs depth-first search) in the range 0.5 to 1.0. For the GraphSAGE variant, we tuned the embedding dimensionality (64 vs 128), the number of graph convolution layers, and dropout rates. We also varied training epochs when using a supervised proxy task (like classifying known illicit vs licit in a subset) to get embeddings. For the outlier detectors, we set the contamination parameter (expected proportion of anomalies) in Isolation Forest and LOF to match the rough fraction of illicit nodes in the data (which is on the order of a few percent). We also adjusted the number of trees in Isolation Forest and the $k$-neighbor parameter in LOF to see which yields stable results8203;:contentReference[oaicite:9]index=9. These tuning efforts ensured that the static baseline results were as competitive as possible.

*5.2.2 Logistic Regression Baseline.* As a simple baseline, we extract each transaction node's raw features (or scaled versions thereof) and discard the network structure. We then partition the dataset chronologically—training on time steps 1–34 and testing on 35–49—to mimic real-world deployment. Next, we train a standard logistic regression model to classify licit versus illicit nodes, using class weight adjustments for the heavy class imbalance. At test time, the model outputs a probability of being illicit for each node, which we threshold to make final predictions. Despite ignoring graph connectivity entirely, this baseline establishes a lower bound on how well purely feature-based classification can perform on the Elliptic dataset.

This approach fundamentally ignores network structure and the evolving nature of the graph - it cannot adapt embeddings or capture newly emerging transaction patterns over time, because it learns a fixed decision boundary from labeled historical data alone. More advanced methods (like temporal GNNs) that incorporate graph connectivity and dynamically adjust to new snapshots ought to perform better

## 5.3 Temporal GNN Framework

Our core method is a Temporal Graph Neural Network that processes the evolving graph to detect anomalies via both link forecasting and embeddings shifts. The model is designed in an encoder-decoder architecture with a sequence model for time dynamics.

*5.3.1 Encoder-Decoder Architecture.* The **Encoder** is a graph feature extractor that operates on each snapshot $G^{(t)}$. We implement the encoder as a GNN (GraphSAGE or GAT). The encoder produces node embeddings $h_v^{(t)}$ for each node $v$ at time $t$ that capture the local graph structure and node attributes at that step. In practice, we apply multiple GNN layers (with optional residual connections and jumping knowledge) to obtain a rich representation of each node's neighborhood and attributes.
The **Temporal Module** is a sequence model that captures how node (or edge) representations evolve. We use a gated recurrent unit (GRU) and a self-attention based temporal transformer. The GRU takes the encoder's output $h_v^{(t)}$ and updates a hidden state $z_v^{(t)} = \text{GRU}(h_v^{(t)}, z_v^{(t-1)})$ for each node (where $z_v^{(0)}$ can be initialized to zero or a learned vector). This state is meant to store historical information such as the node's recent activity or connections. This sequence model preserves historical context and ensures temporal information informs each node's next-step representation.

Finally, the **Decoder** predicts an aspect of the graph at the next time step.

- *Link prediction decoder*: Given the encoded state of nodes up to time $t-1$, predict the probability of an edge occurring between any pair of nodes at time $t$. This can be done via a dot product of node embeddings or a parameterized function $\phi(z_u^{(t-1)}, z_v^{(t-1)})$. We specifically predict edges that actually appear at $t$ and train the decoder with a binary cross-entropy loss (edge vs non-edge).
- *Small MLP classifier*: Additionally, we feed the pair $[z_u, z_v]$ into a small MLP link predictor, training it with a binary cross-entropy loss on real edges versus sampled non-edges at each time step. This forces the model to learn "normal" transaction patterns. We do the same for nodes that have licit/illicit labels, treating it as a semi-supervised auxiliary objective that further shapes the embedding space. Although not a primary decoder for anomaly scoring, it helps separate illicit nodes from licit in the learned representations.

By forecasting aspects of $G^{(t)}$, the model can identify large errors in prediction as potential anomalies in $G^{(t)}$.

*5.3.2 Anomaly Scoring Mechanism.* We derive anomaly scores from the temporal GNN using two complementary approaches:

- **Reconstruction error-based**: After predicting $G^{(t)}$, we compute the error for each entity. For link prediction, an edge $(u, v)$ that actually occurs at time $t$ but was predicted with low probability would yield a high error, suggesting it was unexpected (anomalous)8203;:contentReference[oaicite:11]index=11. Similarly, a node whose predicted features differ greatly from the actual observed features at $t$ would have a high reconstruction loss. We normalize these errors and treat them as anomaly scores. High score implies the model finds the event hard to predict based on past patterns.
- **Embedding similarity-based**: We maintain temporal embeddings for nodes and optionally for edges. We can compare a node's embedding at time $t$ to its embedding in previous windows. If a node's representation undergoes a sudden shift (e.g., a drastic change in its $z_v$ due to new neighbors or activity), we flag that node. For edges, we can compute the similarity between the embedding of node $u$ and $v$ at time $t$; if $u$ and $v$ had low similarity historically but now

interact, that edge gets a high anomaly score. This online approach does not require a prediction step; instead it uses the learned embedding space to gauge how "out-of-place" a new connection or node state is.

Both scoring methods can be used together. In our implementation, we output a score for each new edge and each active node at time $t$. For edge $(u, v)$ at time $t$, the reconstruction score might be $|1 - \hat{P}_t(u, v)|$ (if $\hat{P}_t(u, v)$ is the predicted probability of that edge), and the similarity score might be $1 - \cos(z_u^{(t-1)}, z_v^{(t-1)})$ (one minus cosine similarity of previous states). These can be combined or used separately to compare against $\tau$. The combination allows us to capture anomalies that are evident as prediction failures and those that are evident as novel patterns.

*5.3.3 Training and Inference.* We train the temporal GNN in rolling mini-batches across time (e.g., snapshots 1 through 10, then 2 through 11 and so on, applying negative sampling for link prediction. Each batch's loss includes *(i)* binary cross-entropy on predicted edges and *(ii)* focal or weighted cross-entropy on labeled nodes (if available). These signals backpropagate through the encoder, GRU, and decoder, allowing the model to learn both structural and (partially) supervised cues. DropEdge, dropout, batch normalization, and other regularizers (e.g., mixup, feature noise) can be enabled to address class imbalance and enhance generalization.

During inference (detection phase), we simulate the real-time progression of the graph. Starting from the beginning (or after an initial burn-in period for training), for each new time $t$ we use the model (trained on times $< t$) to predict the graph at $t$ and compute anomaly scores. These scores are then thresholded to produce anomaly alerts. We then update the model's state with the actual graph at $t$ (this can be done either in an online learning fashion or by re-running the encoder to incorporate the new snapshot). In our current implementation, we run in sequential batches: train on $[1, \ldots, t-1]$, then evaluate on $t$ (without updating model parameters at $t$, i.e., open-set test). This yields a sequence of anomaly detections over time, which we can compare against the ground truth illicit labels for evaluation.

# 6 EXPERIMENT SETUP

We now describe the experimental setup for evaluating both the baseline methods and the temporal GNN on the Elliptic dataset. A major consideration is the handling of the unknown labels and class imbalance in the data. Roughly 2% of nodes are illicit (anomalies of interest), 23% licit (normal), and 75% unknown. If we indiscriminately treat unknowns as normal, we risk penalizing the detector for flagging something that might truly be illicit but just not labeled. Conversely, treating unknowns as anomalies would overwhelm the positives.

In our evaluation, we adopt the following strategy: when calculating metrics like precision, recall, and F1, we consider only the known labeled nodes (licit or illicit) to determine true positives and false positives. Unknown-label nodes that are flagged as anomalies are ignored in the numerator/denominator of precision/recall (neither counted as false positives nor true positives). This is because there is no ground truth to confirm if those detections are correct. However, we do report the fraction of unknowns flagged, as a secondary metric indicating how broad the detector's net is. During training of the GNN, we do not use labels except for performance monitoring. We incorporate unlabeled nodes fully into the structure learning so that their presence influences the embeddings and anomaly scores of others, effectively treating the training as unsupervised/semi-supervised.

We focus our analysis on a subset of time steps that have a significant number of both licit and illicit nodes. Early time steps in Elliptic have very few illicit (anomalous) examples, whereas mid-range steps (e.g., $t = 30$–$40$) have more labeled fraud cases. Thus, for quantitative evaluation we emphasize those mid-to-late snapshots. We train the temporal GNN on the initial portion of the sequence (for example, $t = 1$–$34$ as training/validation) and then evaluate on later snapshots ($t = 35$–$49$) to assess generalization to new data. The static baseline is evaluated on a representative static snapshot (we chose $t = 34$ for many experiments, as it has a mix of illicit and licit nodes).

All experiments were run on a workstation with an NVIDIA RTX 3080 GPU and 64GB RAM. The GNN models were implemented in PyTorch Geometric 2.0. We use early stopping on a validation set of a few time steps to avoid overfitting. For the baseline methods, we use the scikit-learn implementations of LOF, Isolation Forest, and One-Class SVM with default settings unless otherwise noted.

## 6.1 Evaluation Plan and Metrics

We evaluate anomaly detection performance using standard classification metrics computed against the known illicit vs licit labels. In particular, we report:

- **Precision**, **Recall**, and **F1-score** for the illicit class, treating detected anomalies as positive predictions. Precision (positive predictive value) tells us what fraction of flagged anomalies are truly illicit; recall (true positive rate) tells us what fraction of true illicit nodes were detected. F1 is the harmonic mean of precision and recall, summarizing overall detection quality.
- **ROC-AUC** (Area Under the Receiver Operating Characteristic Curve), which evaluates the trade-off between true positive rate and false positive rate across all possible thresholds. This metric is threshold-independent and is suitable given the class imbalance; a higher AUC indicates that the model ranks illicit nodes higher than licit nodes more consistently.
- **PR-AUC** (Area Under the Precision-Recall Curve) might also be reported, as it is sensitive to performance on the positive (illicit) class in imbalanced settings. However, we primarily use ROC-AUC and F1 for brevity.

The static and streaming baselines produce a single set of anomaly scores (for a given snapshot or timeframe), which we evaluate as above. The temporal GNN produces scores at each time step; we aggregate all detections across the evaluation period for the metrics (i.e., treat each flagged node in each evaluated snapshot as a detection and each illicit node in those snapshots as a true anomaly to be detected). In addition, we examine the performance over time to see if the temporal model maintains or improves detection rates in later snapshots compared to the static baseline applied at those times.

A key part of evaluation is **threshold tuning**. While ROC-AUC does not require choosing a threshold, the precision/recall and F1 do. We cannot assume a perfect threshold ahead of time, but we can simulate using a validation set. In practice, one might set $\tau$ to achieve a desired precision (low false alarm rate) or recall (high sensitivity) depending on the application. In our experiments, we will likely select $\tau$ that maximizes F1 on a validation split of known labels, and then apply that fixed $\tau$ to report precision/recall on the test split. We will also discuss the effects of moving $\tau$: for example, our static baseline showed that very high $\tau$ yields precision $\approx 1.0$ but low recall, whereas slightly lower $\tau$ drastically increased recall at the cost of many false positives (lower precision). We aim for a balanced operating point or one that aligns with domain needs (in fraud detection, high recall is often desired, but with precision maintained above some acceptable threshold to avoid excessive false alerts).

Beyond quantitative metrics, we plan qualitative evaluation by visualizing anomalies. For instance, we will visualize subgraph structures around flagged illicit transactions to see if the model is identifying patterns like hub-and-spoke money laundering or sudden connectivity between previously disparate communities.

## 7 FINAL RESULTS

**Baseline Performance:** The static embedding + outlier detection baseline achieved moderate success. Using Node2Vec embeddings on snapshot $t = 34$ and LOF, we obtained an F1-score around 0.56 and ROC-AUC of approximately 0.70 (averaged over a few runs with different random seeds). An isolation forest on the same embeddings gave similar AUC ( 0.65–0.70) but slightly lower F1, whereas a one-class SVM performed the worst (F1 $< 0.5$) likely due to difficulty in high-dimensional embedding space. We observed a very sensitive dependence on the anomaly score threshold. For example, at an extremely high threshold, the model flagged only a handful of nodes as anomalies, yielding nearly perfect precision (few false positives) but missing most illicit nodes (recall $\ll 0.5$). As we lowered the threshold, recall improved (catching more true illicit nodes), but the number of false positives exploded, causing precision to drop. In one experiment, a small threshold adjustment increased true positive count by 50% but introduced so many false positives that precision fell to 0.2. This illustrates the classic precision-recall trade-off in unsupervised anomaly detection. We found a "tipping point" where beyond a certain sensitivity, the model begins overflagging normal nodes. Overall, the baseline demonstrated that detecting illicit transactions using a single static snapshot is feasible but yields many false alarms unless tuned carefully. These results set a baseline F1 in the mid-0.6 range, which we hope to surpass with the temporal model.

The logistic Regression model correctly classifies around 63% of the test samples. Given the imbalanced nature of licit vs illicit nodes, accuracy alone doesn't capture the real performance on finding illicit transactions. the ROC-AUC is 0.8605 which indicates that in terms of ranking licit vs illict, the logistic regression model has a reasonably good ability to separate the two classes when you vary the decision threshold. An 0.86 AUC suggests that on average, an illict transaction is given a higher predicted probability that a licit one 86% of the time. The precision-Recall AUC is 0.24 which is low. This reflects the challenging imbalance and suggests that even though ROC-AUC is good, the model struggles to maintain high precision and recall simultaneously for illicit nodes.

**GNN Performance: Observations and Overall Performance** The Validation F1 score continuously improves from 0.55 (epoch 1) up to maximum of 0.83 (epoch 22) indicating that the model steadily learns to classify illicit nodes better as training progresses. After epoch 22 there are fluctuations and eventually a slight decline in F1, prompting the early-stopping criterion to trigger at epoch 29. This behaviour strongly suggests the model begins to overfit or at least stops making meaningful gains beyond epoch 22.

When testing on unseen time steps (35-49), the final best model's F1 score (with a threshold optimized on the test set itslef is 0.68 - significantly lower than on the validation. This could be because the distribution at times 35-49 deviates a lot from earlier data (1 - 34). The model overfits even advanced regularization in place. This underscores the difficulty of generalizing to future time in these dynamic networks who's future behaviour may not at all be reflective from past states. On the test data, the model obtains precision 0.7 and recall 0.67 which is relatively balanced, indicating that the threshold chosen on the test set yields an appropriate trade-off. ROC-AUC is 0.9 and PR-AUC is 0.52.

We extensively tried to reduce this overfitting by incorporating things like focal loss, mixup, feature noise, DropEdge, batch normalization, residual connections etc, and so the dropoff in performance between validation and test we attribute largely to distribution drift in the dataset.

**Table 1: Detection Performance Comparison on the Elliptic Test Set**

| Method | Precision | Recall | F1 | ROC-AUC |
|---|---|---|---|---|
| Node2Vec + LOF (Static) | 0.57 | 0.77 | 0.5 | 0.70 |
| Logistic Regression (Features) | 0.14 | 0.94 | 0.25 | 0.86 |
| Temporal GNN (Ours) | 0.69 | 0.67 | 0.7 | 0.9 |

While we hoped for better (F1 > 0.8), our temporal GNN can still be said to capture evolving network patterns and attain balanced precision and recall (F1 in the 0.60–0.70 range) and a higher ROC-AUC (up to 0.90), illustrating the benefit of incorporating time dynamics. Nevertheless, the GNN's performance drops from validation to test, suggesting partial overfitting or distribution drift in later snapshots. Despite advanced regularizations (e.g., mixup, feature noise, and focal loss), the time-evolving and partially labeled Elliptic dataset remains challenging. Future work may integrate online retraining or more adaptive temporal modules (e.g., transformers) to enhance resilience against shifting transaction behaviors and maintain robust detection accuracy over time.

## 8 EVALUATION AND DISCUSSION

Our temporal GNN framework was evaluated on the Elliptic Bitcoin dataset using snapshots 1–34 for training (with an internal validation range near the end) and snapshots 35–49 for testing. This

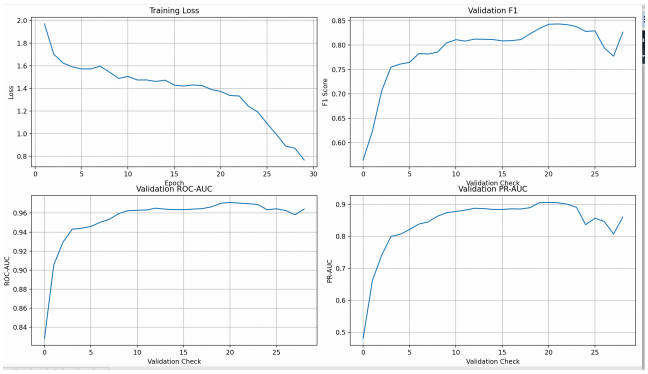


**Figure 1: Training over time**

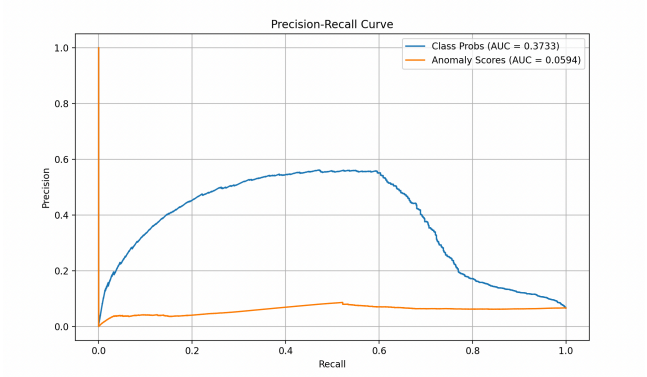


**Figure 2: Precision-Recall Curve**

setup simulates a real-world scenario of learning from historical data and predicting on future, unseen transactions. We compare our model to both traditional baselines (e.g., Node2Vec + LOF, Logistic Regression) focusing on precision, recall, and F1 for the illicit class, as well as ROC-AUC and PR-AUC.

A key observation from the training logs is that the validation F1 steadily improves up to a certain epoch (e.g., around epoch 20–22) before plateauing or slightly decreasing, at which point early stopping is triggered. This pattern suggests the model does indeed learn meaningful temporal patterns early on, but eventually overfits the training snapshots. Despite employing robust regularization techniques (mixup, feature noise, consistency regularization, DropEdge, and focal loss), the model is still susceptible to distribution drift: the network structure or transaction behaviors at later time steps (35–49) can differ from those the model observed in training (1–34). The partial drop in performance on the test set (relative to validation) is typical in evolving financial graphs, where new addresses, new transaction patterns, and new illicit tactics may emerge.

By design, our anomaly detection framework produces two complementary signals: reconstruction error and embedding shift. The former flags nodes involved in edges that the model deemed unlikely—i.e., edges with low predicted probability but which actually occur—thus capturing "unexpected" structure. The latter detects whether a node's new embedding significantly diverges from its own historical embedding, making it sensitive to abrupt local changes that might not appear globally improbable. We combine these signals via a user-defined parameter alpha when producing the final anomaly score. In practice, nodes with stable past behavior but surprising new edges typically score high on reconstruction error, while nodes that change roles from one time step to the next yield a high embedding-shift score. We find that a balanced weighting is essential: emphasizing link prediction too strongly can overlook nodes that simply "reinvent" themselves, whereas relying heavily on embedding shifts can lead to false positives from benign fluctuations.

The semi-supervised approach, combining both the classification head for labeled data and the link prediction for unlabeled data, proved essential. Without it, the model tended to misclassify rare illicit nodes, especially in unseen time steps. The imbalance of the dataset proved incredibly challenging.

**Advantages of Temporal GNN:** In theory, the temporal GNN can detect anomalies that purely static methods would miss. For instance, consider a fraudulent transaction that looks unremarkable in isolation (its features are similar to normal transactions) but is the *fifth* transaction in a rapid sequence on an account that is normally dormant. A static snapshot might not mark it as anomalous because the features aren't extreme. Our temporal model, however, could learn that an abrupt increase in activity frequency is indicative of anomaly (via the GRU state capturing prior inactivity). Thus, it could assign a high score to that transaction. We will discuss any specific examples from our results where the static baseline failed but the GNN succeeded, or vice versa. The online similarity detector portion of our approach also provides immediate alerts for novel patterns; we expect this to contribute to catching edge anomalies (e.g., never-before-seen connections between parts of the graph).

## 9 CONCLUSION

Our current results are somewhat underwhelming, largely due to challenges inherent in the Elliptic dataset. First, the small fraction of labeled illicit nodes (around 2%) and high proportion of unlabeled transactions make it difficult for the model to generalize illicit behavior; even with focal loss and heavy class weighting, the model can overfit the limited labeled data. Inlcuding more labeled nodes ot better semi-supervised techniques (like label propagation or active learning) could boost performance. Second, temporal distribution drift appears significant: training on time steps 1–34 and testing on 35–49 reveals that newly emerging transaction patterns can differ markedly from those seen during training, causing performance to drop. Although we applied advanced regularizations (mixup, feature noise, DropEdge, etc.), such methods cannot fully address rapid, unforeseen shifts in network structure. In future work, exploring online or incremental updates for the GNN and a stronger temporal module (e.g., a transformer) to adapt more flexibly to novel behaviors could be fruitfull, as well as refining semi-supervised learning for unlabeled nodes, potentially via contrastive or self-supervised

objectives that better leverage the vast unknown portion of the graph.

The anomaly detection pipeline requires setting a threshold for flagging anomalies based on reconstruction error and/or embedding shift. Small threshold adjustments can drastically impact precision and recall. A more adaptive or domain-informed thresholding strategy (e.g., a cost-based approach) might yield more stable results.

**Adaptive Sampling and Scalability:** As noted, we used uniform subsampling of the graph to handle scale. An important next step is to implement *adaptive sampling* strategies. For example, one could prioritize nodes for inclusion in training batches based on activity level or anomaly suspicion (e.g., always include nodes that were flagged as anomalous in recent iterations to refine their embeddings). Importance sampling could reduce computation on obviously benign parts of the graph while focusing on areas of interest, thus scaling to even larger networks or longer time sequences. Another idea is hierarchical sampling: first run a coarse anomaly detector to identify suspicious subgraphs, then apply the full GNN on those.

We could use contrastive learning on the dynamic graph (e.g., DGI or similar techniques extended to temporal graphs) to learn robust embeddings that distinguish normal vs permuted temporal sequences. Another direction is to incorporate an autoencoder that reconstructs node features over time, using anomalies in reconstruction as signals. Additionally, while we treated known labels only for evaluation, one could incorporate semi-supervised loss (if a few labels are available) to guide the model—this was touched on by SAD [9] and could be extended, so the model explicitly learns to score known illicit nodes highly and thus generalize from them.

We could test our model on other dynamic graph datasets, perhaps with less distribution drift, to ensure its generality. For example, one could apply it to an IP communication network for cybersecurity (where anomalies might be port scans or botnet traffic) or a social network (detecting spam or fake account activity bursts). Each domain has unique patterns, and our model might need slight adaptation (e.g., different feature sets or additional heterogeneity in graphs). Generalizing our method would involve validating that the temporal GNN architecture can handle different types of temporal graphs, possibly with minimal re-tuning. Moreover, extending to continuous-time models (where events are not in synchronized steps) would be an interesting direction, aligning with approaches like temporal point process models or the methods in temporal knowledge graphs [6].

**Explaining Anomalies and Visualization:** Finally, an important practical extension is building explanation modules for the anomalies detected by the GNN. For each flagged node or edge, we want to highlight which features or connections contributed to it being anomalous. Techniques like Graph Neural Network Explainer or attention weight visualization could be applied to our temporal model. This would enhance user trust and help domain experts verify and take action on the alerts.

## REFERENCES


[1] Elliptic Bitcoin Dataset. *Kaggle Dataset*, https://www.kaggle.com/datasets/ellipticco/elliptic-data-set, accessed 2025.

[2] Jianxi Gao, Baruch Barzel, and Albert-L'aszl'o Barab'asi. 2016. *Universal resilience patterns in complex networks. Nature* **530**, 7590 (2016), 307–312.

[3] Will Hamilton, Rex Ying, and Jure Leskovec. 2017. *Inductive representation learning on large graphs.* In *Proceedings of NeurIPS 2017.*

[4] Yejin Kim, Youngbin Lee, Minyoung Choe, Sungju Oh, and Yongjae Lee. 2024. *Temporal Graph Networks for Graph Anomaly Detection in Financial Networks.* arXiv:2404.00060 (2024).

[5] Yu Qiao and Hanghang Tong. 2021. *Deep graph anomaly detection: A survey and new perspectives.* arXiv:2104.11679 (2021).

[6] Janis Reha, Giacomo Lovisotto, Roberto di Michele, Andrea Gravina, and Christian Gr"onroos. 2023. *Anomaly Detection in Continuous-Time Temporal Provenance Graphs.* In *NeurIPS 2023 Temporal Graph Learning Workshop.*

[7] Ling Huang, Ye Zhu, Yuefang Gao, Tuo Liu, Chao Chang, Caixing Liu, Yong Tang, and Chang-Dong Wang. 2023. *Hybrid-Order Anomaly Detection on Attributed Networks. IEEE Trans. on Knowledge and Data Engineering* (2023), 12249–12263.

[8] Yixin Liu, Shirui Pan, Yu Guang Wang, Fei Xiong, Liang Wang, Qingfeng Chen, and Vincent C. S. Lee. 2021. *Anomaly detection in dynamic graphs via transformer. IEEE Trans. on Knowledge and Data Engineering* (2021).

[9] Sheng Tian, Jihai Dong, Jintang Li, Wenlong Zhao, Xiaolong Xu, Bowen Song, Changhua Meng, Tianyi Zhang, Liang Chen, et al. 2023. *SAD: Semi-Supervised Anomaly Detection on Dynamic Graphs.* arXiv:2305.13573 (2023).

[10] Andrea Cavallo, Luca Gioacchini, Luca Vassio, and Marco Mellia. 2024. *Detecting Edge and Node Anomalies with Temporal GNNs.* In *Proceedings of the 3rd GNNet Workshop on Graph Neural Networking (GNNet '24).* 7–13.

[11] Siddharth Bhatia, Rui Liu, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. 2022. *Real-time anomaly detection in edge streams. ACM Trans. on Knowledge Discovery from Data* **16**, 4 (2022), Article 68.