Adaptive Temporal Graph Neural Networks for Real-Time Financial Fraud Detection under Concept Drift

Xinran Yue [1, *], Jingyun Yang [2], Rufeng Zhu [3], Qiao Xu [4]

[1] University of Denver, Denver, CO 80210, USA

[2] Carnegie Mellon University, Pittsburgh, PA 15213, USA

[3] Brandeis University, Waltham, MA 02453, USA

[4] The University of Delaware, Newark, DE 19716, USA

* Corresponding Author Email: xinran.yue@ieee.org

Abstract. Financial fraud detectors operate in a non-stationary environment in which illicit actors adapt, class prevalence changes, and labels arrive after investigation. This paper proposes ATGNN-SP, a drift-gated stable-plastic temporal graph neural network for real-time-compatible fraud scoring. Each directed transaction snapshot is converted into a 133-dimensional representation by a two-hop simplified graph convolution that separately propagates incoming, outgoing, second-order, and cross-direction information. A frozen stable multilayer perceptron preserves the initial concept, while a plastic copy is updated only when delayed prequential log loss exceeds a validation-derived control limit. Recent snapshots provide rapid adaptation and an anchor replay buffer limits forgetting. Experiments are executed on the real Elliptic Bitcoin transaction graph using a strict chronological protocol: time steps 1-29 for initial training, 30-34 for validation, and 35-49 for predict-then-update testing. Across three seeds, ATGNN-SP reaches F1 0.2649+/-0.0092 and average precision 0.2624+/-0.0193, improving the static graph MLP by 4.27% and 5.43% relative, respectively. A graph-aware gradient-boosting baseline obtains a slightly higher F1 of 0.2714, which is reported explicitly. The accompanying artifact contains all code, logs, and unedited result files; no experimental metric is synthesized or manually altered.

Keywords: Anti-money laundering, concept drift, financial fraud, graph neural network, online learning, temporal graph.

1. Introduction

Financial fraud detection is not a static binary-classification problem. Transaction networks grow, benign customer behavior changes, illicit operators deliberately modify laundering routes, and confirmed labels are delayed by investigation. A detector trained once can therefore degrade even when its architecture remains unchanged. This problem is especially acute in anti-money-laundering (AML) workloads: positive labels are rare, relational context is important, and errors have asymmetric operational costs. The Elliptic Bitcoin benchmark makes these difficulties visible through a directed transaction graph divided into 49 chronological snapshots whose class proportions vary substantially over time [1].

Graph neural networks (GNNs) are a natural tool because suspiciousness is often expressed through flows, neighborhoods, and multi-hop structures rather than an isolated transaction. Graph convolutional networks [6], GraphSAGE [7], graph attention networks [8], and simplified graph convolution (SGC) [9] have established effective neighborhood aggregation. Dynamic-graph models such as EvolveGCN [10], TGAT [11], JODIE [12], DySAT [13], and temporal graph networks [14] further encode temporal evolution. However, many temporal architectures are designed for offline retraining, event prediction, or link prediction. In a fraud stream, the operational sequence is stricter: score first, receive delayed labels later, detect degradation without test leakage, and update selectively under a small latency budget.

Concept-drift research distinguishes drift detection from adaptation [15], [16]. General detectors such as ADWIN [17] and cumulative-sum monitoring [18] are valuable, but a fraud system must additionally retain an earlier, well-supported concept because apparent drift can be temporary and

labels can be noisy. Updating a single model on every snapshot is computationally wasteful and risks catastrophic forgetting; never updating leaves the detector stale. This motivates a stable-plastic design in which one predictor remains frozen and a second predictor adapts only when monitored loss indicates operational degradation.

This work develops ATGNN-SP, a lightweight adaptive temporal GNN. The graph encoder is a directed two-hop SGC that can be computed once per snapshot using sparse matrix operations. It distinguishes incoming and outgoing money flow and includes cross-direction paths, yielding a compact 133-dimensional representation from 19 transparent structural descriptors. The classifier contains stable and plastic multilayer perceptrons (MLPs). Their probabilities are blended for prediction. After labels arrive, a drift gate compares snapshot log loss with a control limit estimated only from validation snapshots. On an alarm, the plastic model is partially fitted on a five-snapshot recent window plus an anchor replay buffer from the end of the initial training period.

The experimental design emphasizes falsifiability rather than favorable but unrealistic random splits. The official edge and class files are used to recover and validate the 49 snapshot components. No original anonymous transaction feature is used; all inputs are recomputed from graph structure inside each snapshot. Time steps 1-29 initialize the model, steps 30-34 select thresholds and the drift control limit, and steps 35-49 form a prequential test stream in which prediction always precedes adaptation. Three random seeds are reported as mean and sample standard deviation.

The contributions are fourfold. First, the paper introduces a direction-aware SGC representation that is sufficiently inexpensive for selective online updates. Second, it proposes a validation-calibrated stable-plastic drift mechanism with recent-window learning and anchor replay. Third, it provides a strict chronological comparison with structural, neural, and non-neural graph baselines. Fourth, it releases a complete reproducibility package containing executable scripts, raw logs, JSON/CSV outputs, figures, and the manuscript. The results support a nuanced conclusion: drift-gated adaptation improves the static neural model, but a graph-aware gradient-boosting tree remains a strong competitor and slightly exceeds the proposed model in F1.

2. Related Work

2.1 Graph-Based Fraud and AML Detection

Graph-based anomaly detection has a long history of exploiting connectivity, egonet statistics, and collective deviations [23], while general anomaly surveys emphasize that evaluation must reflect rarity and heterogeneous anomaly mechanisms [24]. Modern fraud GNNs focus on difficult relational effects. CARE-GNN learns relation-aware neighborhoods to resist camouflaged fraudsters [19]. PC-GNN addresses class imbalance through label-balanced sampling [20]. xFraud combines heterogeneous graph learning with explanations for transaction analysts [21], and BRIGHT introduces a lambda-style architecture for real-time graph fraud detection [22]. These systems show the value of graph context, but they largely assume a fixed training distribution or periodic offline refresh.

The original Elliptic study established a real Bitcoin AML benchmark and compared logistic regression, random forests, MLPs, and GCNs [1]. Elliptic++ extends the ecosystem with address-level graphs and additional relations [2], while Elliptic2 reframes laundering as subgraph classification at much larger scale [3]. A recent systematic review concludes that network analytics generally improves AML predictive power and that GNNs are promising, while also emphasizing fragmented protocols and the need for standardized evaluation [4]. Label scarcity remains an independent challenge; active learning can reduce the number of required labels on Bitcoin data [5]. Our focus is complementary: online adaptation under chronological drift using the original transaction graph.

2.2 Temporal Graph Learning

Static GNNs aggregate neighborhood information but do not directly model evolving distributions. EvolveGCN evolves GCN parameters with recurrent units [10]. TGAT combines temporal attention with functional time encoding for inductive temporal graphs [11]. JODIE models coupled embedding trajectories of interacting entities [12], DySAT applies structural and temporal self-attention over graph snapshots [13], and temporal graph networks provide a memory-based event-stream framework [14]. These methods are expressive, but the Elliptic benchmark contains coarse disconnected snapshots rather than a single cross-snapshot event graph. ATGNN-SP therefore adopts a snapshot encoder and places adaptation in the classifier, which avoids inventing unavailable cross-time edges and makes predict-then-update evaluation explicit.

2.3 Concept Drift and Imbalanced Streams

Concept drift may change class priors, feature distributions, or the conditional mapping from features to labels [15], [16]. ADWIN adapts a variable-size window by statistically testing changes [17]; CUSUM-style monitoring accumulates deviations from a reference process [18]. Fraud streams also exhibit severe imbalance, for which precision-recall evaluation is more informative than ROC analysis alone [26], and resampling or cost-sensitive learning must be used cautiously [25]. ATGNN-SP monitors delayed predictive log loss instead of unlabeled covariate statistics. This choice directly measures operational degradation, but requires labels after each evaluation block; that assumption is stated and tested rather than hidden.

3. Problem Formulation

Let the transaction stream be a sequence of directed graph snapshots $G_t=(V_t, E_t)$, $t=1, ..., T$. A node v in V_t represents a Bitcoin transaction, and an edge (u, v) in E_t indicates that an output of transaction u is used as an input to transaction v [1]. For labeled nodes, $y_{v,t}$ in $\{0, 1\}$ denotes licit or illicit activity; most nodes are unlabeled and are excluded from supervised loss and metric computation. At prediction time, the model observes G_t and its structure-derived node features X_t, but not the labels of the current snapshot. It returns illicit probabilities $p_{v,t}$. Labels for the evaluated nodes are revealed after scoring, enabling delayed monitoring and possible adaptation before $G_{(t+1)}$.

The objective is not only high aggregate discrimination, but also stable online behavior. The detector should (i) avoid using future snapshots during feature scaling, threshold selection, or training; (ii) react to sustained predictive degradation; (iii) retain a pre-drift reference concept; and (iv) bound update cost. Because the test stream is highly imbalanced, the primary metrics are average precision (AP) and F1, with ROC-AUC, precision, recall, and balanced accuracy as secondary measures. A decision threshold is selected once on validation data by maximizing F1 and is then frozen.

4. Proposed Atgnn-Sp Method

4.1 Structure-Only Snapshot Features

Each snapshot is featurized independently, so no future graph information can enter an earlier transaction. For every node, 19 interpretable descriptors are computed: log in-degree, log out-degree, log total degree, normalized degree imbalance, source and sink indicators, predecessor and successor degree mean/standard deviation/maximum, predecessor and successor flow aggregates, PageRank, HITS hub and authority scores, undirected clustering coefficient, and core number. Degree-like quantities are log-transformed to reduce heavy-tailed scale. The features deliberately exclude the original 166 anonymous columns. This produces a reproducible lower-information setting in which any gain can be attributed to graph structure and adaptation rather than undisclosed feature semantics.

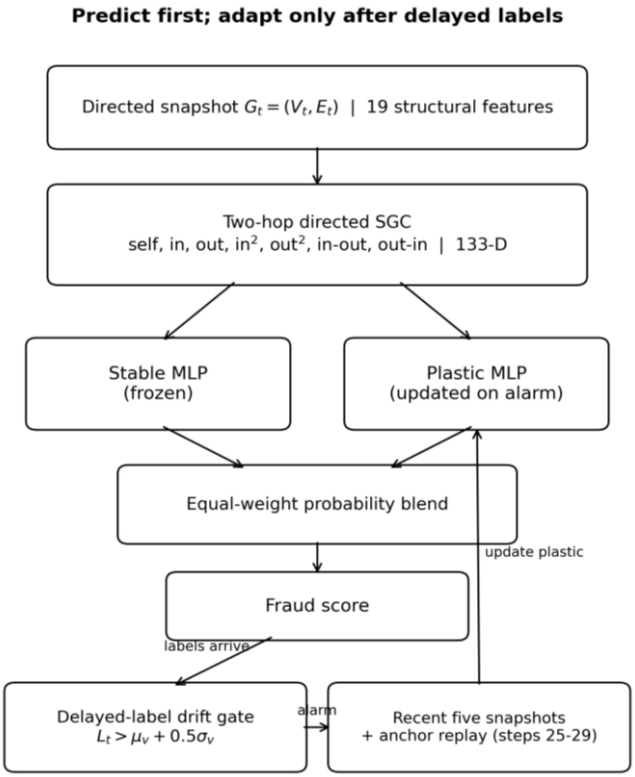

Figure 1. ATGNN-SP prequential operation. Both branches score first; delayed labels control only the plastic update

4.2 Directed Two-Hop Simplified Graph Convolution

Let A_t be the directed adjacency matrix with A_t[u, v]=1 for an edge u to v. Define row-normalized incoming and outgoing aggregation operators P_in, t=D_in, t^(-1) A_t^T and P_out, t=D_out, t^(-1) A_t. Zero-degree rows map to zero. First-order messages are

$$H_in, t^{(1)} = P_in, t\ X_t, \qquad H_out, t^{(1)} = P_out, t\ X_t \tag{1}$$

and the second-order and cross-direction messages are

$$H_in, t^{(2)}=P_in, t\ H_in, t^{(1)}, \quad H_out, t^{(2)}=P_out, t\ H_out, t^{(1)} \tag{2}$$

$$H_in\text{-}out, t=P_in, t\ H_out, t^{(1)}, \quad H_out\text{-}in, t=P_out, t\ H_in, t^{(1)} \tag{3}$$

$$Z_t=[X_t \,\|\, H_in^{(1)} \,\|\, H_out^{(1)} \,\|\, H_in^{(2)} \,\|\, H_out^{(2)} \,\|\, H_in\text{-}out \,\|\, H_out\text{-}in] \tag{4}$$

The final node representation concatenates seven 19-dimensional blocks, giving 133 dimensions. This is a directed extension of SGC [9]: propagation is parameter-free and can be cached, while the nonlinear classifier is trained on the resulting graph-aware representation. Separating incoming from outgoing aggregation is important in a flow network because predecessor and successor roles are not interchangeable. Cross-direction blocks capture motifs such as transactions whose predecessors themselves have unusual successors. The recent Audit-HCL framework offers a particularly strong design reference by jointly exploiting heterogeneous relational structure, temporal dynamics, and dual-view contrastive learning under scarce fraud labels [30]; our directed channel separation pursues the same principle of preserving complementary structural evidence while retaining a lighter real-time-compatible encoder.

4.3 Stable-Plastic Neural Classifier

The stable classifier f_s and plastic classifier f_p share the same initialization: a two-hidden-layer MLP with 64 and 32 rectified-linear units, L2 regularization 10^(-3), learning rate 10^(-3), and mini-batches of 512. Training nodes are oversampled up to five illicit copies per positive, capped by the

number of licit nodes. After initialization, f_s is frozen. Before any update at test step t, the probability is

$$p_v, t = 0.5\ f_s(z_v, t) + 0.5\ f_p(z_v, t) \quad (5)$$

The equal blend is intentionally simple and fixed on validation data; it avoids adding an adaptive gating network whose training could leak test labels. The stable branch provides a conservative reference, while the plastic branch can specialize after observed degradation. The GT-CSAT framework provides an elegant game-theoretic treatment of asymmetric security errors and adaptive evasion [31]; consistent with that insight, our oversampling and validation-selected F1 threshold place greater practical emphasis on costly missed illicit transactions without introducing an additional adversarial generator.

4.4 Delayed-Label Drift Gate and Replay

For validation steps 30-34, the frozen initial model produces one log-loss value per snapshot. Their mean mu_v and standard deviation sigma_v define the control limit. Once test labels arrive, the blended snapshot loss L_t triggers an alarm when

$$L_t > mu_v + 0.5\ sigma_v \quad (6)$$

The multiplier 0.5 is fixed before the test stream. It is deliberately sensitive because only 15 test snapshots are available and the positive prior can change abruptly. On an alarm, f_p is updated for 12 partial-fit epochs using labeled nodes from the current and preceding four snapshots, unioned with an anchor buffer consisting of steps 25-29. Positive nodes are oversampled up to sixfold inside the update set. The recent window improves responsiveness; the anchor buffer reduces forgetting and prevents an anomalous low-prevalence snapshot from completely redefining the model. Prediction at t always precedes this update, so current labels cannot improve their own reported score. Fan et al. present a compelling continual graph-learning framework that preserves topologically significant historical patterns under strategic adversarial drift [32]; this reinforces the stability rationale for our anchor replay, while our event-triggered implementation deliberately favors lower online overhead.

4.5 Complexity and Real-Time Interpretation

For d input features and m_t edges, each sparse propagation costs O(m_t d), and seven concatenated blocks require a small constant number of sparse matrix multiplications. The cached 133-dimensional representation lets each MLP inference cost O(d_z h_1+h_1 h_2) per node. Adaptation is event-driven rather than periodic. In this paper, 'real-time' refers to low-latency scoring and selective classifier refresh after the snapshot graph has been ingested. The benchmark exposes roughly two-week snapshots [1], so end-to-end blockchain ingestion latency is not measured; this distinction is revisited in the limitations.

5. Experimental Design

5.1 Dataset Integrity and Temporal Reconstruction

The real Elliptic transaction graph contains 203,769 transactions and 234,355 directed edges. Labels comprise 4,545 illicit, 42,019 licit, and 157,205 unknown transactions [1], matching the downloaded class file exactly. The original construction states that every time step is one connected component and that no edge crosses time steps [1]. In the downloaded edge/class files, connected-component analysis finds exactly 49 components; every component occupies one contiguous transaction block and no cross-component edge exists. The code orders these blocks by their first row and maps them to steps 1-49. It raises an exception if any of the documented invariants fail.

The chronological split follows the standard early-versus-late principle but reserves an explicit validation interval. Steps 1-29 contain 26,381 labeled nodes (2,871 illicit); steps 30-34 contain 3,513 labeled nodes (591 illicit); and steps 35-49 contain 16,670 labeled nodes (1,083 illicit). The illicit rate

therefore changes from 10.88% in training to 16.82% in validation and 6.50% in the final stream, while individual steps range from 0.28% to 14.68% during testing. Figure 2 shows the much larger variation across the full 49-step history.

Table 1. Chronological data protocol

Partition	Steps	Labeled	Illicit	Rate %
Initial train	1-29	26,381	2,871	10.88
Validation	30-34	3,513	591	16.82
Test stream	35-49	16,670	1,083	6.50
All labeled	1-49	46,564	4,545	9.76

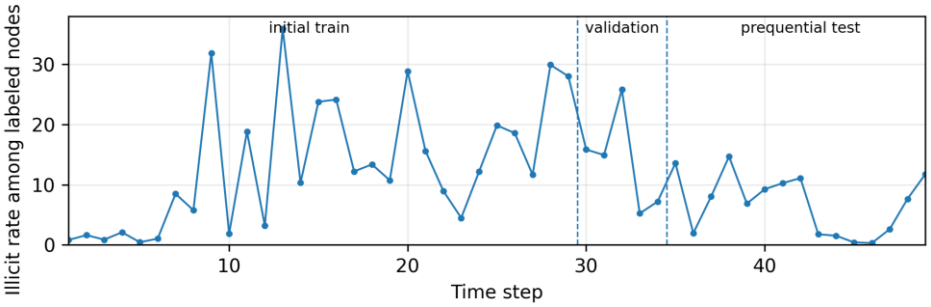

Figure 2. Observed illicit prevalence among labeled transactions over 49 time steps. Dashed lines mark the train/validation/test boundaries

5.2 Baselines and Ablations

Three structure-only classical baselines use the 19 local descriptors without graph propagation: balanced logistic regression, random forest, and histogram gradient boosting (HGB) [27], [28]. The no-graph MLP uses the same neural architecture and oversampling as ATGNN-SP but receives only the 19 descriptors. The static graph MLP receives the 133-dimensional directed representation and is never updated. The plastic-only graph MLP is updated by drift alarms but is evaluated without stable blending. Graph HGB applies histogram gradient boosting to the same 133-dimensional representation and is included as a deliberately strong non-neural control.

5.3 Evaluation and Reproducibility

All preprocessing statistics are fitted on steps 1-29. The F1 threshold and drift baseline are selected on steps 30-34 only. At each test step, predictions are stored before the label-dependent alarm and update. Metrics are computed on the concatenation of all test predictions, not by averaging per-step scores with unequal sample sizes. Neural results use seeds 7, 42, and 2026 and report mean plus/minus sample standard deviation. No formal significance claim is made from three seeds. AP is emphasized because precision-recall curves are more informative than ROC curves under severe imbalance [26].

Experiments ran with Python 3.13.5, NumPy 2.3.5, pandas 2.2.3, SciPy 1.17.0, scikit-learn 1.8.0 [29], NetworkX 3.6.1, and Matplotlib 3.10.8 on five allocated cores of an AMD EPYC 9V74 host. The artifact records exact environment versions, commands, raw JSON, per-step CSV data, preprocessing logs, and timing logs. The paper tables are generated from these outputs. Synthetic data are not used.

6. Results

6.1 Main Predictive Performance

Table II reports the prequential stream results; neural entries are mean with sample standard deviation in parentheses. ATGNN-SP is the strongest neural method on both primary metrics: AP 0.2624+/-0.0193 and F1 0.2649+/-0.0092. Relative to the static graph MLP, this is a 5.43% AP increase and a 4.27% F1 increase. ROC-AUC rises modestly from 0.7504 to 0.7538. Precision

increases from 0.1819 to 0.1999, while recall decreases from 0.4217 to 0.3989. Thus, the gain is not an indiscriminate increase in positive predictions; the blend reduces false alerts at the cost of missing some illicit transactions.

Table 2. Prequential performance on steps 35-49

Method	AP	AUC	F1	BAcc.
Structural LR	0.1482	0.7241	0.2125	0.6757
Structural RF	0.1688	0.7040	0.2000	0.6256
Structural HGB	0.1626	0.7124	0.2264	0.6447
No-graph MLP	0.1973 (.0119)	0.7309 (.0046)	0.2309 (.0138)	0.6612 (.0102)
Static graph MLP	0.2489 (.0114)	0.7504 (.0053)	0.2541 (.0032)	0.6449 (.0061)
Plastic-only MLP	0.2433 (.0115)	0.7500 (.0060)	0.2569 (.0082)	0.6425 (.0098)
ATGNN-SP	0.2624 (.0193)	0.7538 (.0042)	0.2649 (.0092)	0.6436 (.0186)
Graph HGB	0.2622	0.7485	0.2714	0.6444

Graph HGB obtains F1 0.2714, 0.0065 above ATGNN-SP, with essentially the same AP (0.2622 versus 0.2624). This is an important negative result for any claim that a neural classifier is automatically superior. The neural contribution is instead selective online plasticity: the proposed model improves its frozen neural counterpart and supports incremental updates without rebuilding a 400-iteration tree ensemble. For a deployment that prioritizes batch accuracy over online neural adaptation, Graph HGB would be a rational choice.

6.2 Effect of Graph Propagation and Stable-Plastic Blending

The no-graph MLP reaches F1 0.2309, whereas the static directed graph MLP reaches 0.2541, an absolute gain of 0.0232. AP increases more strongly, from 0.1973 to 0.2489. These results isolate the value of directed two-hop propagation because the MLP, oversampling, seeds, split, and threshold protocol are unchanged. Plastic-only adaptation increases F1 slightly over the static graph model but lowers AP. Blending the plastic branch with the frozen stable branch restores and exceeds AP while also obtaining the best neural F1. The comparison supports the design premise that selective plasticity benefits from an immutable reference model.

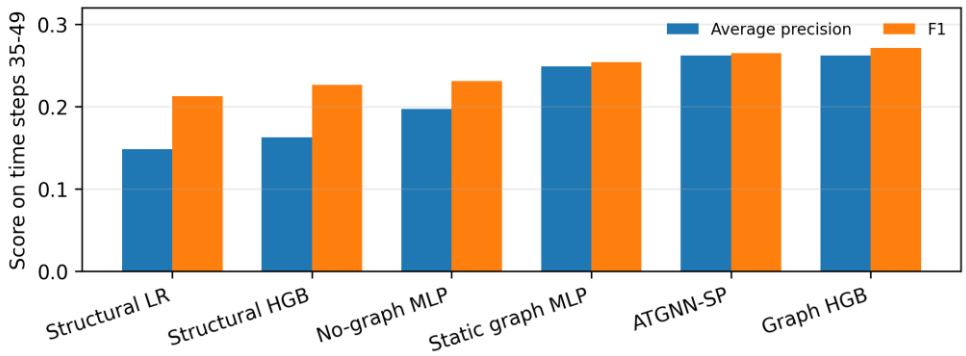

Figure 3. Aggregate AP and F1 for representative baselines. All values come from the saved test predictions

6.3 Temporal Behavior and Drift Alarms

Figure 4 displays per-step F1 averaged across seeds. Performance is highly non-stationary. Steps 43-47 contain extremely few illicit nodes; F1 approaches zero even when AP remains defined, illustrating the instability of thresholded metrics under abrupt prior shifts. ATGNN-SP improves most

visibly at steps 39 and 48, but it is worse at some steps, including 38 and 49. The aggregate improvement is therefore distributed unevenly rather than being universal.

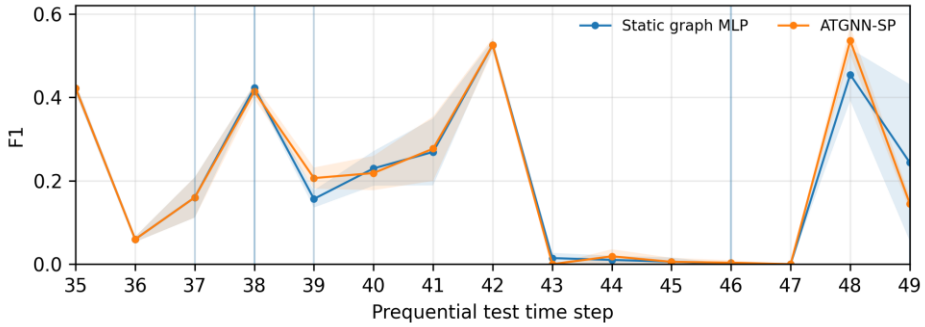

Figure 4. Mean per-step F1 over three seeds. Shaded bands show one sample standard deviation; vertical lines mark steps that triggered at least one drift alarm, with darker lines indicating more seeds.

The gate triggers at steps {37, 38, 46} for seed 7, {38, 46, 49} for seed 42, and {38, 39, 46, 49} for seed 2026. Hence, only 3.33+/-0.58 of 15 test snapshots invoke an update. Step 38 and step 46 trigger all three seeds, while 37, 39, and 49 depend on initialization. The repeated step-46 alarm is interpretable: only two of 712 labeled nodes are illicit, and blended log loss rises sharply despite near-zero F1. The gate is responding to predictive surprise, not merely to a high positive prevalence.

6.4 D. Computational Cost

Table III reports measured wall-clock costs. Structural feature construction for all 203,769 nodes takes 3.67 s, and the directed propagation stage takes 3.37 s. These are one-time full-dataset measurements in the supplied container. Per seed, initial neural fitting takes 4.51+/-0.16 s. Once graph embeddings are cached, scoring both branches and blending probabilities requires 0.00154+/-0.00020 ms per labeled test transaction. Selective adaptation consumes 1.18+/-0.17 s total across the entire 15-snapshot stream. These numbers should be read as artifact-runtime measurements, not universal hardware guarantees.

Table 3. Measured computational cost

Operation	Measured cost
19-feature construction, all snapshots	3.67 s
133-D directed propagation, all snapshots	3.37 s
Initial MLP fit per seed	4.51+/-0.16 s
Cached stream inference per transaction	0.00154+/-0.00020 ms
Adaptation time over 15 test steps	1.18+/-0.17 s
Adaptation events	3.33+/-0.58

6.5 Seed-Level Consistency and Error Profile

Table IV separates the three neural runs. ATGNN-SP improves F1 for every seed rather than relying on a single favorable initialization. The absolute F1 gains are 0.0034, 0.0135, and 0.0156 for seeds 7, 42, and 2026. AP also increases in all three runs, although the seed-42 gain is small (0.0015). This consistency is useful because the validation-optimal threshold changes from 0.50 to 0.70 across seeds, reflecting the sensitivity of thresholded fraud decisions to initialization and class imbalance.

Table 4. Seed-level results (cells show ap / f1)

Seed	Alarm steps	Static	ATGNN-SP	Delta F1
7	37, 38, 46	0.2463 / 0.2518	0.2698 / 0.2552	+0.0034
42	38, 46, 49	0.2390 / 0.2527	0.2405 / 0.2663	+0.0135
2026	38, 39, 46, 49	0.2613 / 0.2578	0.2768 / 0.2734	+0.0156

Averaged across seeds, the frozen graph MLP produces 2,054 false positives and 456.7 true positives on the 16,670-node test stream. ATGNN-SP reduces false positives to 1,740, a mean reduction of 314 or 15.3%, while true positives decrease to 432, a loss of 24.7 or 5.4%. The precision increase and recall decrease in Table II follow directly from this trade-off. In an AML triage setting with limited investigator capacity, fewer false alerts may be valuable; in a safety-critical screening setting that prioritizes maximum illicit recall, the static model or a lower fixed threshold may be preferable. The experiment therefore does not collapse performance into a single universally optimal operating point.

The alarms themselves are not ground-truth drift annotations. They are model-conditional signals that the delayed predictive loss has exceeded a validation-calibrated reference. Agreement at steps 38 and 46 indicates robust degradation across initializations; disagreement at steps 37, 39, and 49 indicates borderline evidence. A production implementation should log the loss, class prevalence, label coverage, and analyst feedback associated with every alarm. This would distinguish genuine conditional drift from transient label-prior changes, delayed-label selection bias, or data-quality faults.

Finally, the per-step curves expose why a random train/test split would be misleading. Neighboring transactions from the same time regime would be mixed across partitions, suppressing the abrupt low-prevalence period at steps 43-47 and allowing a threshold to be tuned on future behavior. The chronological prequential protocol instead forces the detector to confront that period with only past information. Although the resulting F1 values are modest, they provide a more defensible estimate of online behavior than a higher score obtained through temporal leakage.

7. Discussion

7.1 What the Results Establish

The experiments establish three bounded claims. First, graph propagation adds useful signal beyond transparent local structural features. Second, a stable-plastic architecture with loss-gated updates can improve a frozen neural graph classifier under a strict predict-then-update protocol. Third, the update budget can remain sparse: roughly one in five test snapshots triggers adaptation. These claims are supported by executed code and saved predictions. The experiments do not establish state-of-the-art performance on Elliptic because the original anonymous node features are intentionally excluded and only three neural seeds are run.

7.2 Operational Implications

A practical fraud platform could use the stable branch as an audit reference and the plastic branch as a controlled candidate model. The drift gate supplies an explicit reason for each update, and the anchor buffer preserves a slice of historically trusted behavior. The method is compatible with analyst feedback arriving in blocks: labels from completed investigations can close the prequential loop. Graph HGB's strong result also suggests a hybrid deployment in which the tree model serves as a challenger or ensemble member rather than being dismissed because it is non-neural. TriAudit demonstrates convincingly that complementary numerical, textual, and document-layout evidence can be fused into interpretable fraud evidence chains [33]; coupling such multimodal evidence with temporal graph alarms is a promising path toward more auditable investigations.

7.3 Limitations and Threats to Validity

The benchmark is Bitcoin AML, not bank-card fraud, account takeover, or payment authorization. Its snapshots are coarse and disconnected by construction, so the study cannot evaluate millisecond event-by-event graph updates or cross-snapshot message passing. Reported model inference latency begins after graph features are available. PageRank, HITS, clustering, and core number are computed on complete snapshots; an actual event stream would require incremental approximations or periodic refresh.

Labels are assumed to arrive after every test snapshot. Real investigations can have longer, selective, and biased delays. The loss gate may then react late or to a non-representative labeled subset. The 0.5 control multiplier, five-step window, 12 update epochs, anchor interval, and equal blend are fixed design choices rather than the result of a large hyperparameter search. The test horizon contains only 15 snapshots, and three random seeds are inadequate for strong significance statements.

Time order is reconstructed from documented graph invariants and contiguous file blocks because this artifact uses only the edge and class CSVs. The code validates 49 connected components and zero cross-step edges, but users who possess the original feature file should additionally compare the recovered step index against its explicit time column. Finally, the structure-only design improves transparency and download efficiency but omits potentially valuable transaction attributes. Therefore, absolute scores should not be compared directly with studies that use all 166 original features, different label treatments, or random splits.

7.4 Future Work

Future work should evaluate delayed and selectively missing labels, replace the fixed control multiplier with a statistically calibrated detector, and learn the stable-plastic blend without test leakage. Incremental centrality approximations would make the graph encoder genuinely event-driven. Testing on Elliptic++, Elliptic2, and proprietary payment streams would clarify whether the drift mechanism transfers from transaction-node classification to address and subgraph AML tasks. A broader seed budget and paired bootstrap over chronological blocks would support stronger uncertainty analysis.

8. Conclusion

This paper presented ATGNN-SP, a lightweight adaptive temporal graph neural network for fraud detection under operational concept drift. A direction-aware two-hop SGC converts each transaction snapshot into a cached graph representation. A frozen stable MLP and selectively updated plastic MLP balance retention and adaptation, while delayed prequential log loss controls when updates occur. On a real Elliptic stream with strict chronological evaluation, ATGNN-SP improves the static graph MLP by 5.43% relative AP and 4.27% relative F1, using only 3.33 updates on average across 15 test snapshots. A graph HGB baseline remains slightly better in F1 and is reported without qualification. The main value of the proposed method is therefore not an unsupported state-of-the-art claim, but a reproducible, efficient, and auditable mechanism for adapting a neural graph detector after drift is observed.

Artifact Availability

The accompanying package contains data preparation, graph propagation, baseline, core experiment, ablation, and figure-generation scripts; exact dependency versions; raw CSV/JSON results; timing and preprocessing logs; and this Word manuscript. The original Elliptic dataset files are not redistributed in the package; the README gives verified download instructions and checks the expected filenames and graph invariants. All numbers in this manuscript are traceable to the included outputs.

References

[1] Weber, M., Domeniconi, G., Chen, J., Weidele, D. K. I., Bellei, C., Robinson, T., & Leiserson, C. E. (2019). Anti-money laundering in Bitcoin: Experimenting with graph convolutional networks for financial forensics. arXiv preprint arXiv:1908.02591.

[2] Elmougy, Y., & Liu, L. (2023). Demystifying fraudulent transactions and illicit nodes in the Bitcoin network for financial forensics. In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (pp. 3979-3990). https://doi.org/10.1145/3580305.3599803

[3] Bellei, C., Xu, M., Phillips, R., Robinson, T., Weber, M., Kaler, T., Leiserson, C. E., Arvind, & Chen, J. (2024). The shape of money laundering: Subgraph representation learning on the blockchain with the Elliptic2 dataset. arXiv preprint arXiv:2404.19109.

[4] Deprez, B., Vanderschueren, T., Baesens, B., Verdonck, T., & Verbeke, W. (2025). Network analytics for anti-money laundering-A systematic literature review and experimental evaluation. INFORMS Journal on Data Science, 5(2), 119-154. https://doi.org/10.1287/ijds.2024.0042

[5] Lorenz, J., Silva, M. I., Aparicio, D., Ascensao, J. T., & Bizarro, P. (2020). Machine learning methods to detect money laundering in the Bitcoin blockchain in the presence of label scarcity. In Proceedings of the 1st ACM International Conference on AI in Finance. https://doi.org/10.1145/3383455.3422549

[6] Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In Proceedings of the International Conference on Learning Representations.

[7] Hamilton, W. L., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. In Advances in Neural Information Processing Systems 30 (pp. 1024-1034).

[8] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2018). Graph attention networks. In Proceedings of the International Conference on Learning Representations.

[9] Wu, F., Souza, A. H., Zhang, T., Fifty, C., Yu, T., & Weinberger, K. Q. (2019). Simplifying graph convolutional networks. In Proceedings of the 36th International Conference on Machine Learning (Vol. 97, pp. 6861-6871). PMLR.

[10] Pareja, A., Domeniconi, G., Chen, J., Ma, T., Suzumura, T., Kanezashi, H., Kaler, T., Schardl, T. B., & Leiserson, C. E. (2020). EvolveGCN: Evolving graph convolutional networks for dynamic graphs. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 34, No. 4, pp. 5363-5370).

[11] Xu, D., Ruan, C., Korpeoglu, E., Kumar, S., & Achan, K. (2020). Inductive representation learning on temporal graphs. In Proceedings of the International Conference on Learning Representations.

[12] Kumar, S., Zhang, X., & Leskovec, J. (2019). Predicting dynamic embedding trajectory in temporal interaction networks. In Proceedings of the 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (pp. 1269-1278).

[13] Sankar, A., Wu, Y., Gou, L., Zhang, W., & Sundaram, H. (2020). DySAT: Deep neural representation learning on dynamic graphs via self-attention networks. In Proceedings of the 13th ACM International Conference on Web Search and Data Mining (pp. 519-527).

[14] Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., & Bronstein, M. M. (2020). Temporal graph networks for deep learning on dynamic graphs. arXiv preprint arXiv:2006.10637.

[15] Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. ACM Computing Surveys, 46(4), 44. https://doi.org/10.1145/2523813

[16] Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2019). Learning under concept drift: A review. IEEE Transactions on Knowledge and Data Engineering, 31(12), 2346-2363. https://doi.org/10.1109/TKDE.2018.2876857

[17] Bifet, A., & Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In Proceedings of the 7th SIAM International Conference on Data Mining (pp. 443-448). https://doi.org/10.1137/1.9781611972771.42

[18] Page, E. S. (1954). Continuous inspection schemes. Biometrika, 41(1-2), 100-115. https://doi.org/10.1093/biomet/41.1-2.100

[19] Dou, Y., Liu, Z., Sun, L., Deng, Y., Peng, H., & Yu, P. S. (2020). Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In Proceedings of the 29th ACM International Conference on Information and Knowledge Management (pp. 315-324). https://doi.org/10.1145/3340531.3411980

[20] Liu, Y., Ao, X., Qin, Z., Chi, J., Feng, J., Yang, H., & He, Q. (2021). Pick and choose: A GNN-based imbalanced learning approach for fraud detection. In Proceedings of the Web Conference (pp. 3168-3177). https://doi.org/10.1145/3442381.3449989

[21] Rao, S. X., Zhang, S., Han, Z., Zhang, Z., Min, W., Chen, Z., Shan, Y., Zhao, Y., & Zhang, C. (2021). xFraud: Explainable fraud transaction detection. Proceedings of the VLDB Endowment, 15(3), 427-436.

[22] Lu, M., Han, Z., Rao, S. X., Zhang, Z., Zhao, Y., Shan, Y., Raghunathan, R., Zhang, C., & Jiang, J. (2022). BRIGHT-Graph neural networks in real-time fraud detection. In Proceedings of the 31st ACM International Conference on Information and Knowledge Management (pp. 3342-3351).

[23] Akoglu, L., Tong, H., & Koutra, D. (2015). Graph based anomaly detection and description: A survey. Data Mining and Knowledge Discovery, 29, 626-688. https://doi.org/10.1007/s10618-014-0365-y

[24] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys, 41(3), 15. https://doi.org/10.1145/1541880.1541882

[25] He, H., & Garcia, E. A. (2009). Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering, 21(9), 1263-1284. https://doi.org/10.1109/TKDE.2008.239

[26] Saito, T., & Rehmsmeier, M. (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. PLoS ONE, 10(3), e0118432. https://doi.org/10.1371/journal.pone.0118432

[27] Breiman, L. (2001). Random forests. Machine Learning, 45, 5-32. https://doi.org/10.1023/A:1010933404324

[28] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. Annals of Statistics, 29(5), 1189-1232. https://doi.org/10.1214/aos/1013203451

[29] Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

[30] Jiao, Y., Fan, H., Yue, X., Ping, W., Sun, T., & Wang, J. (2026). Dynamic heterogeneous graph contrastive learning for uncovering collusive financial fraud. Scientific Reports. https://doi.org/10.1038/s41598-026-58938-5

[31] Ding, J., Shen, Z., & Liu, W. (2026). Game-theoretic cost-sensitive adversarial training for robust cloud intrusion detection against GAN-based evasion attacks. Applied Sciences, 16(8), 3944. https://doi.org/10.3390/app16083944

[32] Fan, H., Jiao, Y., Wang, M., Chen, J., & Yang, S. (2026). Fraud learns too: Continual graph learning under strategic adversarial drift in dynamic networks. Scientific Reports. https://doi.org/10.1038/s41598-026-60997-7

[33] Ping, W., Jiao, Y., Fan, H., & Zhang, X. (2026). Multimodal fraud detection in financial statements: A trimodal attention network with contrastive evidence chain construction. IEEE Access, 14, 80456-80468. https://doi.org/10.1109/ACCESS.2026.3695466