International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 11 | Nov - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

A Graph Neural Network Platform for Blockchain Anti-Money Laundering: A Literature Review and Proposed System Architecture

1st Prof. Shahid R. Siddiqui
Guide
Dept. of Computer Engineering
Sandip Institute of Technology and Research Centre
Nashik, Maharashtra, India
shahid.siddiqui@sitrc.org

2nd Abhishek Shirsath
Student
Dept. of Computer Engineering
Sandip Institute of Technology and Research Centre
Nashik, Maharashtra, India
shirsathabhi512@gmail.com

3rd Sanjivani Ughade
Student
Dept. of Computer Engineering
Sandip Institute of Technology and Research Centre
Nashik, Maharashtra, India
sanjeevanidughade16@gmail.com

4th Shubham Kasar
Student
Dept. of Computer Engineering
Sandip Institute of Technology and Research Centre
Nashik, Maharashtra, India
shubhamkasar8099@gmail.com

5th Jidnyasa Shinde
Student
Dept. of Computer Engineering
Sandip Institute of Technology and Research Centre
Nashik, Maharashtra, India
Jidnyasashinde5@gmail.com

***Abstract*—Cryptocurrency networks give criminals a fast, borderless and pseudonymous way to launder money, yet the public ledgers that make this possible also produce transaction graphs far too large and too fast-moving for manual or rule-based review. Graph neural networks (GNNs) have become the leading modelling approach for this problem because they work directly on the account-transaction graph and not on flattened, per-transaction feature vectors. This paper reviews the recent literature on GNN-based blockchain anti-money laundering (AML) and organizes eighteen studies into five threads: foundational models built on the Elliptic benchmark, attention-based and heterogeneous architectures, temporal and ensemble methods, explainability and privacy-preserving techniques, and early system-level platforms for real-time use. A comparative analysis and a six-point gap analysis show that few studies deal with end-to-end platforms that combine incremental graph maintenance, low-latency inference and investigator-facing explanation. To fill this gap, the paper proposes ChainSentry-GNN, a six-layer reference architecture covering ledger ingestion, incremental graph construction, model serving, explainability, adaptive thresholding and an analyst feedback loop, together with an evaluation methodology centred on detection quality, latency, throughput and cross-chain generalization.**

***Index Terms*—Graph neural networks, blockchain, anti-money laundering, cryptocurrency forensics, transaction graph, real-time inference, explainable AI, Elliptic dataset.**

I. INTRODUCTION

Cryptocurrency has changed how value travels. A transfer that once needed a chain of correspondent banks, several days and a stack of paperwork can now be completed in minutes, between two people who have never met and who may sit in different legal systems. That convenience is the reason digital assets are useful, and it is also the reason they appeal to anyone who needs to hide where money came from. Money laundering, in its simplest definition, is the effort to make the proceeds of crime look like ordinary, lawful funds. Public blockchains give launderers speed, global reach and pseudonymous addresses, all of which make that effort easier than it is in a conventional bank.

There is an irony here, though. A bank keeps its ledger private, so only the institution (and its regulator) can see who paid whom. Most public blockchains do the opposite: every transfer ever made is stored in a ledger that anyone can download. The data are pseudonymous rather than anonymous, and in principle each hop of a laundering route can be followed. In practice the volume is the obstacle. Public networks record enormous numbers of transfers, and a human analyst clicking through an explorer one address at a time cannot keep up. Fixed rules such as “flag anything above a given amount” are easy to learn and easy to evade, so launderers simply split their activity to stay beneath them.

Researchers have therefore started treating blockchain laundering as a graph problem. Wallets and addresses become nodes, transfers become directed edges, and the familiar laundering tricks (layering funds through many intermediaries, forwarding value along peeling chains, pushing coins through mixing services) show up as recognizable shapes in that graph. Graph neural networks (GNNs) are well matched to this view because they build the representation of a node out of its neighbourhood, including neighbours several hops away, instead of judging each transaction in isolation as a row of engineered features. Since the release of the Elliptic dataset [1], graph-based models have repeatedly done better than conventional tabular classifiers on the same data, and the field has since grown quickly: attention mechanisms, temporal encoders, self-supervised pre-training, ensembles, diffusion-based graph recovery and federated training have all been tried [2]–[20].

Most of that work, however, stops at the benchmark. A typical paper trains on a graph that was extracted once from a historical dump, reports precision and recall on a held-out set, and ends. What it rarely describes is the part that a compliance team actually lives with: new blocks arriving every few seconds or minutes, a graph that has to be kept current without being rebuilt, a scoring service with a bounded response time, and an alert that an investigator can read, challenge and either clear or escalate. The distance between a good number on a benchmark and a monitoring service that can be run every day is the motivation for this paper.

We therefore review the literature with a platform lens. Our contributions are the following:

- a structured review of eighteen studies on GNN-based blockchain AML, grouped into five threads, with a comparison table covering approach, focus and data domain;
- a comparative and critical analysis of what the field does well, where the methodology is weak, and six concrete research gaps that follow from it;
- ChainSentry-GNN, a six-layer reference architecture that joins incremental graph maintenance, directed heterogeneous attention scoring, typology-aligned explanation, reinforcement-learning-based thresholding and an analyst feedback loop; and
- an evaluation methodology that measures system behaviour (latency, throughput, explanation fidelity and cross-chain transfer) alongside detection quality.

The remainder of the paper is organized as follows. Section II gives the background needed to read the rest of the paper. Section III reviews the literature. Section IV compares the studies, Section V discusses strengths and limitations, and Section VI sets out the research gaps. Section VII describes the proposed architecture, Section VIII the evaluation methodology, Section IX regulatory and ethical considerations, Section X limitations and future scope, and Section XI concludes.

II. BACKGROUND AND PRELIMINARIES

A. How Laundering Unfolds on a Public Ledger

Anti-money-laundering literature usually describes laundering in three stages. In placement, illicit funds first enter the financial system; in layering, they are moved around to break the link with their origin; and in integration, they re-emerge as seemingly legitimate wealth. On a blockchain, placement often means depositing criminal proceeds (for example ransomware payments or the takings from a scam) into a set of fresh addresses or converting cash into coins through an exchange. Layering is where most of the visible graph activity happens, because it is cheap to create addresses and cheap to send coins through them.

Several recurring typologies are discussed throughout the studies reviewed here. Structuring (sometimes called smurfing) splits a large sum into many small transfers that individually look unremarkable. A peeling chain moves a large balance through a long sequence of addresses, and at each hop a small amount is “peeled” off to a new destination while the remainder continues down the chain. Mixing or tumbling services pool coins from many users and pay out equal-sized portions, which deliberately blurs which input corresponds to which output. Rapid pass-through describes addresses that receive value and forward almost all of it within a very short time, behaving more like a relay than like a wallet. Finally, chain-hopping moves value across blockchains through bridges or exchanges, which leaves a gap in any analysis confined to a single ledger.

B. Blockchain Data Models and Graph Construction

The way a graph is built depends on how the underlying chain keeps its records. Bitcoin follows the unspent-transaction-output (UTXO) model: a transaction consumes earlier outputs and creates new ones, so the natural graph links transactions to transactions through the outputs they spend. Ethereum and similar networks use an account model in which balances sit at externally owned accounts and smart-contract addresses, and a transfer is an edge between two such accounts. Elliptic, for example, represents Bitcoin transactions as nodes and the flow of coins between them as edges [1], while many Ethereum studies treat addresses as nodes [15], [16].

A second modelling decision concerns identity. Many people hold several addresses, and an analysis that treats every address as a separate entity may miss the fact that a dozen of them belong to one actor. Heuristics such as clustering the inputs of one transaction together can recover some of this structure on UTXO chains, but they are imperfect and their errors propagate into any model trained on top. We return to this point when discussing the ingestion layer in Section VII.

C. Formal Graph Setting

We write the transaction graph at time t as $G(t) = (V(t), E(t), X(t))$, where $V(t)$ is the set of nodes (addresses or transactions), $E(t)$ is the set of directed, timestamped edges, and $X(t)$ collects node and edge features. Each node carries a type (for instance ordinary account, contract, exchange-tagged address or mixer) and each edge carries an amount, a fee and a timestamp. Unlike the static graphs common in computer-vision and citation-network research, $G(t)$ grows

continuously as blocks are confirmed. Nodes have a very uneven degree distribution as well: a few exchange or service addresses connect to millions of counterparties, while most addresses have only a handful of neighbours. Both properties matter for the choice of model and for how it is deployed.

Table I collects the notation used in the remainder of the paper, so that the equations in Sections II and VII can be read without hunting for definitions.

TABLE I
SUMMARY OF NOTATION

Symbol	Meaning
G(t) = (V(t), E(t), X(t))	Transaction graph at time t
h_v(l)	Embedding of node v at layer l
N(v)	Neighbours of v (incoming and outgoing)
a_iv(t)	Attention coefficient of neighbour v for transaction i
ŷ_i(t)	Risk score of transaction i at time t, in [0, 1]
τ	Escalation threshold applied to the risk score
w+	Positive-class weight in the training loss
α, β, γ	Reward coefficients with α + β + γ = 1
η, δ, ε	Learning rate, discount factor, exploration rate

D. Graph Neural Network Building Blocks

Almost every model reviewed in this paper is a variation of message passing. In each layer, a node collects messages from its neighbours, combines them, and updates its own representation; after k layers, a node's embedding reflects information from nodes up to k hops away. The architectures differ mainly in how the neighbour messages are weighted and combined.

The graph convolutional network (GCN) uses a fixed, degree-normalized average of neighbour features:

$$H(l+1) = \sigma(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} H(l) W(l)) \quad (1)$$

where $\tilde{A} = A + I$ is the adjacency matrix with self-loops, $\tilde{D}$ its degree matrix, W(l) a trainable weight matrix and σ a non-linearity. It is simple and scales well, but every neighbour is given a weight fixed by degree alone. The graph attention network (GAT) replaces this fixed weighting with learned coefficients:

$$\alpha_{uv} = \exp(e_{uv}) / \Sigma_{k \in N(v)} \exp(e_{kv}), \quad e_{uv} = \text{LeakyReLU}(a^{\top} [W h_u \| W h_v]) \quad (2)$$

so that the model can decide that one neighbour matters more than another. For AML this is an attractive property, since a single suspicious counterparty should count for more than ten routine ones. GraphSAGE learns an aggregation function over a sampled neighbourhood, which makes it inductive, meaning that it can embed nodes that were not present during training:

$$h_v(l+1) = \sigma(W \cdot [h_v(l) \| \text{AGG}(\{ h_u(l) : u \in N(v) \})]) \quad (3)$$

The graph isomorphism network (GIN) uses a sum aggregator followed by a multilayer perceptron, which gives it expressive power comparable to the Weisfeiler–Lehman graph-isomorphism test:

$$h_v(l+1) = \text{MLP}((1 + \epsilon) h_v(l) + \Sigma_{u \in N(v)} h_u(l)) \quad (4)$$

Table II summarizes how these models compare for the AML task. None of them is best on every axis, which is one reason ensemble methods [14] and heterogeneous variants [9], [10] have attracted attention.

TABLE II
COMPARISON OF CORE GNN LAYERS FOR AML

Model	Aggregation	Strength	Caveat for AML
GCN	Degree-normalized mean	Simple, scalable	Equal-ish weight to all neighbours; assumes undirected edges
GAT	Learned attention	Can emphasise risky neighbours; weights are inspectable	Higher cost; attention is not automatically a faithful explanation
GraphSAGE	Sampled neighbourhood	Inductive; suits a growing graph	Sampling may miss rare illicit neighbours
GIN	Sum + MLP	High discriminative power	Sensitive to hub nodes and degree
Hetero. / directed	Type- and direction-specific	Encodes account types and money-flow direction	More parameters; needs typed data

E. Benchmarks and Evaluation Measures

The Elliptic dataset contains roughly 204 thousand Bitcoin transactions linked by about 234 thousand payment-flow edges and spread over 49 time steps [1]. Each node has 166 features, of which 94 describe the transaction itself and 72 summarize its neighbours. Only a minority of nodes are labelled: about 2% are marked illicit, about 21% licit, and the rest are unknown. Elliptic2 moves from nodes to subgraphs and is built for the case where the laundering pattern is a group of addresses rather than a single one [2].

The extreme imbalance makes plain accuracy a poor measure; a model that always predicts "licit" can reach a high accuracy while detecting no crime at all, which is exactly the trap Lawal et al. highlight [18]. The literature therefore relies on precision, recall and F1 computed for the illicit class, on the area under the precision–recall curve (PR-AUC), and occasionally on the Matthews correlation coefficient. Because the data are temporal, a time-ordered split (train on earlier steps, test on later ones) is the more honest protocol, since a random split lets the model peek at the future.

F. Feature Families Used in the Literature

Whatever the architecture, a GNN is only as informative as the features it starts from, so it helps to know what the

benchmark studies actually feed in. Elliptic supplies two kinds of node feature: local attributes of the transaction itself, such as time step, number of inputs and outputs, fees and amounts, and aggregated attributes summarizing the same quantities over the transaction's one-hop neighbourhood [1]. The second group is, in effect, a hand-made version of what a GNN learns on its own, which partly explains why graph models gain relatively little from it and why raw-feature baselines are surprisingly competitive on this dataset.

Beyond Elliptic, studies on account-based chains and on bank data tend to add three further families. Behavioural features capture how an account acts over time: dormancy before a burst of activity, inter-arrival times between transfers, and the ratio of value received to value sent. Structural features describe the position of a node in the graph, such as degree, clustering and distance to known tagged entities. Type features record whether an address is a contract, an exchange deposit address or a known service. Heterogeneous models [8], [10] rely heavily on the last family, which is one reason their transfer to untagged pseudonymous chains is not guaranteed.

G. From Rules to Graphs: Why the Modelling Paradigm Matters

It is worth being explicit about what graph models replace. Traditional transaction monitoring relies on hand-written rules (a transfer above a limit, many transfers in a day, a link to a blacklisted address). Rules are transparent and easy to audit, and that is why regulators are comfortable with them. Their weakness is that they are static and local: a rule sees one transaction, or at most one account, and a patient launderer who knows the rule can stay just below it. Every new evasion technique then needs a new rule, written by hand and tested against a long tail of false alarms.

Tabular machine-learning classifiers improve on this by learning a decision boundary from labelled examples, but they still treat each record as independent. Whatever the model learns about relationships has to be pre-computed into features by an engineer, which is how the aggregated features of Elliptic came about. GNNs move one step further by learning those relational features themselves, end to end, and can in principle discover patterns that no one thought to encode. The price is a loss of the simple transparency that rules provide, which is why the explainability layer in Section VII is not a decoration but a condition for use.

H. Cost-Sensitive Thinking in AML Evaluation

A further peculiarity of AML is that errors are not equally expensive. A missed illicit transaction can bring regulatory penalties and reputational damage, while a false alarm costs analyst time. Analyst time is limited, however, and a queue flooded with false positives becomes useless in practice because real alerts are buried among them. The practical question is therefore never just "how accurate is the model?" but "how many alerts can the team review, and how many of them turn out to matter?". This is the reasoning behind including alert-to-SAR conversion and false-positive rate among our metrics, and behind the cost terms of the reward in Eq. (9).

I. A Small Worked Example of Message Passing

A tiny example makes the idea concrete. Suppose address A sends funds to B, B forwards almost everything to C, and C sends the amount on to D, which is a known mixing service. With a single message-passing layer, B's embedding is updated from A and C, so it now carries a hint that one of its neighbours is a relay. After a second layer, A's embedding has also been informed by C, and B's by D, so the fact that the chain ends in a mixer has propagated back two hops toward the source. A model that looks only at A's own features would never see this, whereas a two-layer GNN encodes it automatically. This also shows why depth matters and why too much depth is harmful: the information on the mixer helps A, but after many more layers it would be diluted into everything else in the graph.

III. LITERATURE REVIEW

We organize the reviewed work into five threads. The first covers foundational graph models built on the Elliptic benchmark. The second covers attention-based, heterogeneous and directed-graph architectures that refine how relational information is aggregated. The third deals with temporal, multi-view and ensemble methods aimed at robustness to evolving typologies. The fourth concerns explainability, robustness and privacy-preserving training, which determine whether a model is usable in a regulated setting. The fifth turns to system-level work that begins to treat deployment as part of the problem.

This review differs from earlier surveys in its emphasis. The systematic review of Deprez et al. [12] looks at network analytics for AML across domains and concentrates on how methods are evaluated. Our own interest is narrower in data (public blockchains) and wider in scope, since we ask what is needed to turn a model into a running service. The studies are therefore read for what they say about data handling, latency and explanation, in addition to what they report about accuracy.

A. Foundational Graph Models on the Elliptic Benchmark

Weber et al. released the Elliptic dataset and, in the same study, showed that graph convolutional networks could outperform tabular baselines such as random forests on the same underlying features [1]. The result was important less for its absolute figures than for what it demonstrated: that the neighbourhood of a transaction carries information that its own attributes do not. Alarab et al. then examined how well GCNs cope with Bitcoin classification specifically [3], and later paired temporal graph convolution with an LSTM so that the way the graph changes from one step to the next informs the prediction, rather than a single frozen snapshot [4].

Label scarcity is the next obstacle. Confirmed illicit examples are rare and expensive to obtain, so Lo et al. proposed Inspection-L, which first pre-trains a Graph Isomorphism Network encoder with a self-supervised Deep Graph Infomax objective and only then attaches a Random Forest classifier [5]. In effect, most of the learning happens without labels. More recently, Bellei et al. pointed out that node- and edge-level labels do not capture how laundering is really organized, since it usually involves coordinated groups of addresses. Their Elliptic2 dataset attaches labels to subgraphs and is large enough to resemble exchange- and bank-scale networks [2].

Taken together, this first thread establishes three ideas that later work builds upon: graph structure beats flat features, self-supervision can reduce the dependence on labels, and the unit of detection may need to be a subgraph rather than a node. Its limitation is that all of it is evaluated on Bitcoin snapshots.

B. Attention-Based, Heterogeneous and Directed-Graph Architectures

A second thread asks how information should be aggregated. Cheng et al. built a group-aware deep graph learning model that is explicitly designed to catch laundering coordinated across several accounts, rather than scoring each account on its own [6]. Li et al. approached a different difficulty with DIGA, a guided diffusion model that recovers missing or deliberately hidden edges in a transaction graph [7]. This targets the connectivity gaps and label scarcity that launderers exploit when they scatter their activity across many addresses.

Johannessen and Jullum extended the message-passing neural network to a heterogeneous graph of account types and transaction relations. Their results suggest that modelling those types explicitly is better than collapsing everything into one homogeneous graph [10]. Direction is another structural fact that is easy to lose. Funds have a source and a destination, yet many spectral GNNs are not designed for directed graphs. Poon et al. addressed this with LineMVGNN, which builds a line graph over edge adjacencies and passes messages in both directions between payment and receipt transactions [9]. Olaniyi et al. combined heterogeneous graph attention with an explainability layer, aiming at multi-layered financial-crime networks that span transaction, device and geographic features at the same time [8].

What unites these studies is the view that the data have more structure than a plain graph. Types, direction, groups and missing edges all carry meaning, and models that respect them tend to do better. The cost is added complexity and, in most cases, an assumption that typed data are available, which is not always true on pseudonymous public chains.

C. Temporal, Multi-View and Ensemble Approaches

The third thread responds to two facts: laundering behaviour changes over time, and every single architecture has blind spots. Chen et al. proposed MDST-GNN, a multi-distance spatial-temporal model that looks at interactions at several hop distances and infers temporal patterns from the anonymized timestamps in public blockchain data. It reports consistent gains over static baselines on Elliptic [13]. Haider et al. took an ensemble route, combining GCN, GAT and GIN predictions through tuned soft voting. They report that the ensemble detects a majority of illicit Elliptic transactions while keeping the false-positive rate under one percent [14].

Ntuala et al. proposed HTFD, which stacks a GCN layer and a GAT layer and adds an economic penalty mechanism. The mechanism does more than score a transaction: it dynamically reduces the on-chain influence of addresses flagged as fraudulent. The model was validated on three large Ethereum stablecoin networks, with throughput and latency among the concerns examined [15]. A related but distinct threat, phishing rather than laundering, was tackled by Li et al. with PDGNN, an end-to-end graph classification framework for Ethereum addresses [16]. Its relevance to AML lies in the shared machinery, since both tasks are about identifying malicious behaviour in an account graph.

Read critically, this third thread shows a trade-off. Every added component (a temporal encoder, an extra architecture in an ensemble, a penalty mechanism) improves some measure, but also adds parameters, training cost and one more thing that must be kept consistent in production. Ensemble results in particular deserve careful reading, because voting over three networks triples

inference cost, and the benefit has to be weighed against the latency budget of a live service, a point the cited studies mostly do not quantify.

D. Explainability, Robustness and Privacy

Compliance decisions have to be defended, so a fourth thread asks whether a GNN's output can be trusted and explained rather than simply being accurate. Lawal et al. studied GNN behaviour under the class imbalance of Elliptic, where illicit transactions make up under two percent of labelled nodes. They showed that impressive training accuracy can hide a model that has effectively learned only the majority class [18]. Marasi and Ferretti compared GCN, GAT and GraphSAGE for cryptocurrency AML under matched conditions, which is one of the few like-for-like benchmarks available [11]. Deprez et al. contributed a systematic literature review of network analytics for AML and catalogued how experiments are typically evaluated [12].

Privacy completes the picture. Banks and exchanges cannot freely pool customer transaction records, yet laundering routes cross institutional boundaries. Kadhe et al. describe a federated learning framework that trains over vertically and horizontally partitioned financial data, so that a payment network and its partner institutions can jointly improve anomaly detection without centralizing raw records [19]. Although this work is not blockchain-specific, the idea transfers naturally to exchanges that each see only part of a laundering route.

A recurring observation in this fourth thread is that the questions it raises are as much about evaluation practice as about modelling. Imbalance, data leakage between training and test periods, and the absence of shared reporting standards make it hard to say whether a reported improvement is real or an artefact of the experimental set-up. For a platform, this matters because the model's expected behaviour in production, not its benchmark score, is what an institution has to defend to a supervisor.

E. Toward Platforms: System-Level and Real-Time Designs

The smallest thread is also the one closest to the aim of this paper, because it treats AML detection as a running system. Khanvilkar and Kommuru propose a real-time monitoring framework that builds a dynamic transaction graph, classifies suspicious behaviour with a GNN, and pairs each flagged case with a retrieval-augmented generation module that writes a natural-language explanation aligned with regulatory clauses [17]. Cui et al., working on general financial fraud rather than laundering, introduced FraudGNN-RL, in which a temporal-spatial-semantic graph encoder is combined with a Deep Q-Network that adapts the escalation threshold as the alert queue and the fraud patterns drift [20]. That design transfers readily to the alerting layer of a blockchain AML platform.

This thread signals that latency, incremental graph maintenance and investigator-facing explanation are beginning to be seen as first-class design requirements. Even so, none of the reviewed studies integrates incremental on-chain ingestion, low-latency GNN inference and compliance-aligned explanation in one documented platform architecture. Table III lists the eighteen studies and their focus.

Two caveats apply to this fifth thread. The retrieval-augmented explanation of [17] is demonstrated in a banking context rather than on a public blockchain, and the reinforcement-learning threshold of [20] targets general financial fraud. Both therefore indicate what is possible and neither amounts to a ready-made blockchain AML platform, which is precisely the space that the architecture in Section VII is meant to occupy.

TABLE III
SUMMARY OF REVIEWED LITERATURE ON GNN-BASED BLOCKCHAIN AML METHODS

Ref	Study & approach	Focus	Data domain
[1]	Weber et al., 2019 – GCN forensic graph model	Illicit node classification (Elliptic)	Bitcoin
[2]	Bellei et al., 2024 – Elliptic2 subgraph learning	Subgraph-level laundering "shapes"	Bitcoin
[3]	Alarab et al., 2020 – GCN competence study	Node classification accuracy	Bitcoin
[4]	Alarab & Prakoonwit, 2022 – Graph-based LSTM	Temporal graph convolution	Bitcoin
[5]	Lo et al., 2022 – Inspection-L (DGI+GIN+RF)	Self-supervised embeddings	Bitcoin
[6]	Cheng et al., 2023 – Group-aware deep graph learning	Coordinated multi-account laundering	Bank transactions
[7]	Li et al., 2023 – DIGA diffusion graph recovery	Graph completion under label scarcity	Bank transactions
[8]	Olaniyi et al., 2026 – GNN–XAI framework	Multi-layered crime network detection	Multi-source
[9]	Poon et al., 2025 – LineMVGNN	Line-graph multi-view message passing	Bank / mobile payments
[10]	Johannessen & Jullum, 2023 – Heterogeneous GNN	Money-launderer node classification	Bank transactions
[11]	Marasi & Ferretti, 2024 – Comparative GNN study	Benchmarking GCN / GAT / GraphSAGE	Bitcoin
[12]	Deprez et al., 2024 – Systematic review	Network analytics for AML	Cross-domain
[13]	Chen et al., 2025 – MDST-GNN	Multi-distance spatial-temporal modelling	Bitcoin
[14]	Haider et al., 2025 – Quantum-ready ensemble GNN	Voting / stacking GCN+GAT+GIN ensemble	Bitcoin
[15]	Ntuala et al., 2025 – HTFD dual-layer GNN	Detection + economic penalty mitigation	Ethereum stablecoins

Ref	Study & approach	Focus	Data domain
[16]	Li et al., 2022 – PDGNN	Phishing-account graph classification	Ethereum
[17]	Khanvilkar & Kommuru, 2025 – Regulatory graph GenAI	Real-time monitoring + LLM explanation	Banking
[18]	Lawal et al., 2025 – Explainable GNN framework	Interpretability under class imbalance	Bitcoin (Elliptic)

F. Synthesis Across the Five Threads

Laid side by side, the five threads describe a progression more than a set of competing schools. The first established that graphs help. The second and third refined what the graph should contain and how it should evolve. The fourth asked whether the output can be trusted, and the fifth began to ask how it would be run. What is missing is not another architecture for the scoring step, since the literature has plenty of those, but the connective tissue between steps: the code, schemas and procedures that let a graph, a model, an explanation and an analyst work as one pipeline. That connective tissue is the subject of Section VII.

IV. COMPARATIVE ANALYSIS

The studies in Table III can be compared along three dimensions. The first is representation: does the model capture direction, node types and multi-hop structure explicitly? The second is the evaluation setting: is the model tested on a static snapshot or on an evolving stream? The third is system-readiness: are latency, throughput and explanation treated as design requirements or as afterthoughts?

A. Representation

On representation, the trend is clear and favourable. Early work used plain GCNs [1], [3]. Later work encodes transaction direction and account type [6]–[10], recovers hidden edges [7], looks at several hop distances [13] and blends several architectures to hedge against the weakness of any one [14]. In effect, the field has moved from asking "does the graph help?" to asking "which structural facts about money flow should the model be told?".

B. Evaluation Setting

The picture on evaluation is less encouraging. Counting the domains in Table III, nine of the eighteen studies work with Bitcoin, five with bank or banking-style transaction data, two with Ethereum, one with multi-source data and one is a cross-domain review. The bulk of the Bitcoin work relies on the same Elliptic snapshot, so conclusions about generality are drawn from a single chain at a single moment in history. Only a small number of studies examine an evolving stream [15], [17], and even then the stream is usually a replay rather than a live feed.

C. System-Readiness

System-readiness is the thinnest area. Latency or throughput are reported by only a few studies [15], [17], and fewer still link the model's output to an explanation that an investigator can use and that remains reliable given the class-imbalance concerns raised by Lawal et al. [18]. No reviewed study combines incremental on-chain graph updates, sub-second inference and investigator-facing, regulation-aligned explanation in one documented architecture. This is the observation that motivates the ChainSentry-GNN design in Section VII.

D. Capability Comparison with Platform-Oriented Work

To make the system-readiness gap concrete, Table IV sets the three platform-oriented studies [15], [17], [20] against the capabilities we consider necessary for a deployable service, and against the architecture proposed in this paper. A dash means that the capability is not addressed in the study as summarized in Section III. The table reflects our own reading of the literature and is not an exhaustive audit of each paper.

TABLE IV
CAPABILITY COMPARISON WITH PLATFORM-ORIENTED STUDIES

Capability	[15]	[17]	[20]	Proposed
Incremental / dynamic graph	–	Partial	Partial	Yes (L2)
Latency / throughput evaluated	Yes	Partial	–	Yes (planned)
Typology / regulatory explanation	–	Yes	–	Yes (L4)
Adaptive alert threshold	–	–	Yes	Yes (L5)
Analyst feedback loop	–	–	Partial	Yes (L6)
Multi-chain design	Partial	–	–	Yes (design)

The pattern is plain: each study contributes one or two of the needed capabilities, and none contributes all of them. The proposed architecture is not meant to outperform these works on their own terms. Its purpose is to show how the capabilities can be combined and evaluated together.

V. DISCUSSION: STRENGTHS AND LIMITATIONS

A. Strengths of the Reviewed Body of Work

The literature makes a convincing case that graph-native models do better than tabular classifiers on the same blockchain data. The advantage is most visible for coordinated, multi-account laundering that is practically invisible when each transaction is scored on its own [1], [6], [13]. Self-supervised and diffusion-based methods show that useful representations can be learned even when confirmed illicit labels are extremely scarce [5], [7]. Ensemble and multi-view designs improve robustness against the noise and sparsity typical of public blockchain

graphs [9], [14]. And the arrival of real-time and explanation-oriented work suggests that the community has started to count deployment constraints as part of the problem rather than as an engineering detail left for later [15], [17].

B. Limitations and Methodological Concerns

Several limitations recur across the papers. First, most studies are benchmarked on Elliptic or Elliptic2. Both are valuable public resources, but each represents a single blockchain at a fixed period, so transfer to other chains or to more recent obfuscation techniques is often assumed rather than shown [1]–[5], [13]. Second, class imbalance is acknowledged almost universally but handled inconsistently, and Lawal et al. demonstrate that a model can look highly accurate while having failed to learn the illicit class at all [18].

Third, studies that report throughput or latency remain the exception [15]. Fourth, explainability is frequently a post-hoc step rather than a design constraint, which weakens the link between the explanation and the typologies investigators are trained to recognize [8], [17]. Fifth, comparisons between papers are difficult, because preprocessing, split protocol and metric choice differ and hyperparameter budgets are seldom reported. Finally, almost none of the studies explain how a deployed model would be retrained as investigator feedback and new confirmed cases arrive.

C. Recurring Technical Challenges

Beyond the limitations listed above, several technical difficulties come up repeatedly and shape the design choices that follow. We mention them briefly here because they explain why a simple "train a GCN and deploy it" recipe is not enough.

Over-smoothing and depth. Stacking many message-passing layers makes node embeddings converge toward one another, which blurs exactly the difference between an illicit node and its licit neighbours. Laundering routes can be long, yet deep networks smooth away the signal, so practical models stay shallow and rely on attention, skip connections or multi-distance designs [13] to reach further without losing resolution.

Heterophily. Standard GNNs assume that connected nodes tend to be alike. In laundering graphs this assumption is only partly true: illicit addresses deliberately transact with ordinary ones, and an illicit node is often surrounded by licit neighbours. Attention and type-aware aggregation help, because they let the model learn which neighbours to trust, but the mismatch remains a reason why simple averaging can underperform.

Noisy and missing labels. Labels come from investigations, law-enforcement actions or exchange tags, so they are late, incomplete and sometimes wrong. The large "unknown" class in Elliptic is a reminder that for most nodes nobody knows the true answer. Semi-supervised and self-supervised training [5] are partial remedies, and positive-unlabelled learning is another plausible direction.

Concept drift. Typologies change as launderers adapt and as tooling changes: a mixing service that is popular this year may disappear next year. A model frozen on historical data will gradually lose accuracy, which is why we place retraining and threshold adaptation inside the architecture rather than treating them as maintenance.

Scale and hubs. Real chains contain hubs with millions of edges. Full-neighbourhood aggregation on such nodes is expensive, and sampling is a necessary compromise, although it risks dropping the very rare edge that carries the signal.

VI. RESEARCH GAP ANALYSIS

Putting the limitations above together yields six gaps, each of which the proposed architecture is designed to address.

Incremental graph maintenance. Most GNN-AML models are trained on a graph extracted once from a historical dump. Few describe how the graph and its embeddings are updated as new blocks arrive without retraining from scratch [1]–[9], [13]. In a live setting this is not optional, because a full rebuild on every block would never keep up.

Latency- and throughput-aware evaluation. Detection quality dominates the reported metrics, while inference latency and sustained throughput are reported by only a small minority [15], [17]. A model that is accurate but cannot score a block before the next one arrives is of limited operational use.

Cross-chain and cross-asset generalization. Validation is concentrated on Bitcoin, with comparatively little work validating the same architectures on Ethereum, stablecoin or Layer-2 transaction graphs [1]–[5], [15], [16]. Laundering rarely stays on one chain, so this gap matters.

Architecture-native explainability. Explanations are typically attached after training instead of being designed in, so the rationale does not reliably correspond to recognized AML typologies [8], [17], [18]. An investigator

who receives a heat-map of attention weights but no statement about which typology is suspected still has to do most of the interpretive work.

Closed-loop retraining. Reward and feedback mechanisms driven by confirmed investigator dispositions are well studied in the adjacent general-fraud literature [20], but they are rarely integrated into blockchain-specific GNN pipelines.

Robustness to adversarial restructuring. Launderers actively change their on-chain behaviour in response to detection, yet few reviewed studies measure how performance degrades under such adversarial drift.

VII. PROPOSED PLATFORM ARCHITECTURE: CHAINSENTRY-GNN

A. Design Requirements

The goal of ChainSentry-GNN is not another GNN model in isolation. It is a system that keeps a live transaction graph current, serves risk scores within bounded latency and produces explanations an investigator can use, while feeding analyst decisions back into the model. From the gaps in Section VI we derive six requirements: R1, chain-agnostic ingestion; R2, delta-based graph updates; R3, a scoring model that respects direction and node type; R4, explanations tied to recognized typologies; R5, thresholds that adapt to analyst workload; and R6, a feedback path that turns dispositions into training signal.

We assume a defender who observes only public on-chain data (plus optional exchange-supplied tags), cannot see off-chain identities and faces an adaptive adversary who can split transfers, route through new addresses and use mixing services. Attacks on the platform's own infrastructure are out of scope.

B. Overview

The platform comprises six layers arranged as a pipeline with a feedback path from the last layer back to the scoring model, as shown in Fig. 1. The first two layers deal with data, the third with prediction, the fourth with human-readable output, and the last two with operations and learning. Table V summarizes the inputs, outputs and main design concern of each layer.

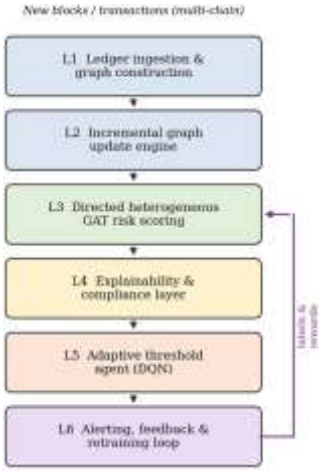


Fig. 1. Layered architecture of ChainSentry-GNN with the analyst feedback path.

TABLE V
LAYER-WISE RESPONSIBILITIES OF CHAINSENTRY-GNN

Layer	Input → Output	Main design concern
L1 Ingestion	Raw blocks → typed, timestamped graph elements	Chain-agnostic schema; address clustering errors
L2 Incremental update	New elements → refreshed rolling graph	Cost proportional to the affected subgraph
L3 Risk scoring	Graph + features → risk score in [0,1]	Direction, node types, class imbalance
L4 Explanation	Flagged cases → typology and rationale	Fidelity to the model's decision
L5 Adaptive threshold	Queue state → escalation threshold	Trade-off of missed cases vs analyst load
L6 Feedback loop	Dispositions → labels and rewards	Label delay; avoiding feedback bias

C. Ledger Ingestion and Graph Construction Layer

A chain-agnostic ingestion service subscribes to new blocks and parses transactions into typed nodes (externally owned accounts, contract addresses, exchange-tagged addresses and, where known, mixing-service addresses) and typed, timestamped, directed edges. Putting a thin adapter in front of each chain, rather than writing separate pipelines, means that Bitcoin-style and account-based chains share one internal schema, which is what makes cross-chain evaluation feasible later.

Node and edge features are engineered from transfer amount, fee, counterparty degree and dormancy, following the feature-construction conventions of the Elliptic-derived literature [1], [3] but generalized across chains and asset types. Optional address-clustering heuristics can merge addresses that are probably controlled by one actor. Because such heuristics can be wrong, the layer records the cluster membership as metadata rather than silently merging nodes, so that a mistaken merge can be undone without rebuilding the graph.

Table VI groups the features computed by this layer. The grouping mirrors the families discussed in Section II and is

kept identical across chains where the underlying concept exists, so that a model trained on one chain can be evaluated on another without redefining its inputs.

TABLE VI
FEATURE GROUPS COMPUTED BY THE INGESTION LAYER

Group	Examples	Purpose
Value	Amount, fee, fee rate	Size and cost of the transfer
Structure	In/out degree, distinct counterparties	Position in the local graph
Temporal	Dormancy, dwell time, inter-arrival time	Burst and relay behaviour
Type flags	Contract, exchange-tagged, mixer-tagged	Entity context where known
Neighbourhood	Mean / max neighbour amount	Cheap aggregate cues
Flow ratio	Value in / value out	Pass-through detection

D. Incremental Graph Update Engine

Instead of re-extracting the whole graph for each scoring pass, the update engine applies delta changes to a rolling graph window as every new block arrives. Embeddings are refreshed only for the affected neighbourhood, in the spirit of the multi-hop modelling used by MDST-GNN [13], so the cost of an update scales with the size of the changed subgraph and not with the length of the chain history. In practice, a new transaction touches its sender and receiver, and its influence is propagated outward for as many hops as the model has layers.

Two engineering choices deserve mention. The rolling window bounds memory by retiring nodes that have been inactive for a configurable period, while archiving a compact summary so that an old address that suddenly reactivates (a dormancy pattern worth flagging) can be reconstructed. High-degree hubs, such as large exchange addresses, are handled by neighbour sampling, in the manner of GraphSAGE, to prevent a single popular node from dominating the update cost.

E. GNN Risk-Scoring Model Layer

The scoring layer is a directed, heterogeneous graph attention network, built on the design principles of directed and line-graph architectures [9], [10]. For a transaction i at time t, the attention coefficient over each incoming or outgoing neighbour v combines the endpoint embeddings, the edge features e_{iv} and the edge direction/type r:

$$a_{iv}(t) = \mathrm{softmax}_v\left(\mathrm{LeakyReLU}\left(a^{\top} \left[W h_i \,\|\, W h_v \,\|\, W_e e_{iv} \right] \right) \right) \quad (5)$$

The neighbourhood embedding is then the attention-weighted sum of type-specific projections of the neighbour embeddings:

$$h_i(t) = \sum_{v \in N(i)} a_{iv}(t) \cdot W_r h_v(t) \quad (6)$$

and the risk score is produced by a sigmoid output layer:

$$\hat{y}_i(t) = \sigma\left(w^{\top} h_i(t) + b \right) \quad (7)$$

where W, W_r, W_e and w are trainable projection parameters, b is a bias, and $\sigma(\cdot)$ maps the result to a continuous risk score in [0, 1] instead of a hard label. Producing a continuous score is deliberate, since the adaptive threshold layer needs it, and it is consistent with the ensemble literature, which supports scoring against more than one trained model variant [14]. A practical deployment may run two or three variants and average their scores.

Because illicit examples are so rare, training uses a class-weighted binary cross-entropy over a mini-batch B:

$$L = -(1/|B|) \sum_{i \in B} \left[w_{+} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \right] \quad (8)$$

with w_{+} chosen from the inverse class frequency or tuned on a validation period. Self-supervised pre-training in the style of Inspection-L [5] can be used to initialize the encoder when labels are especially limited, and training and validation sets are always split chronologically.

F. Explainability and Compliance Layer

Every transaction that crosses the escalation threshold passes through an explanation layer. It maps the attention weights a_{iv} and the contributing subgraph onto a fixed taxonomy of recognized on-chain laundering typologies, following the typology-aligned approach shown by GNN–XAI and regulatory-graph frameworks [8], [17]. It then produces a short natural-language rationale for the investigator. Table VII illustrates the mapping between typologies, the graph signature that triggers them and the evidence surfaced to the analyst.

TABLE VII
TYPOLOGY-ALIGNED EXPLANATION TEMPLATES

Typology	Graph signature	Evidence surfaced
Structuring	Fan-out of many small transfers from one source, later re-converging	Source, split sizes, convergence address
Peeling chain	Long linear path; most value forwarded each hop	Hop sequence, peeled amounts
Rapid pass-through	Value in ≈ value out within a very short dwell time	Dwell time, in/out ratio
Mixing interaction	Short path to a mixer-tagged node; equal-sized outputs	Path, mixer tag, output sizes
Chain-hopping	Bridge or exchange hop followed by activity on another chain	Bridge address, linked chain

Explanations are only useful if they are faithful. The layer therefore reports a fidelity check (does removing the highlighted subgraph actually lower the score?) and, following Lawal et al. [18], evaluates reliability separately on the illicit class, so that an impressive overall figure cannot hide a poor result on the cases that matter.

G. Adaptive Threshold Reward Formulation

A fixed cut-off will be wrong sooner or later: too low and investigators drown in false positives, too high and illicit activity slips through. A reinforcement-learning agent, implemented as a Deep Q-Network in the spirit of FraudGNN-RL [20], observes the current queue state (recent false-positive rate, investigator backlog and score distribution) and selects the escalation threshold τ applied to the scores $\hat{y}_i(t)$. The reward for threshold action a at time t is defined as

$$R_t(a) = -(\alpha \cdot FN_cost(t) + \beta \cdot FP_cost(t) + \gamma \cdot T_review(t)) - P_delay(t) \quad (9)$$

where FN_cost(t) is the estimated regulatory and reputational cost of a missed illicit transaction, FP_cost(t) the estimated investigator time wasted on a cleared false positive, and T_review(t) the mean time to disposition for escalated transactions. The coefficients satisfy $\alpha + \beta + \gamma = 1$, and the penalty P_delay(t) rises when SAR-eligible cases are not filed within the regulatory filing window. The agent updates its action-value function with the standard temporal-difference rule

$$Q(s, a) \rightarrow Q(s, a) + \eta \cdot [R_t + \delta \cdot \max_a' Q(s', a') - Q(s, a)] \quad (10)$$

where η is the learning rate and δ the discount factor applied to the next state s' and its available actions a'. An ε-greedy policy balances exploration of alternative thresholds against exploitation of thresholds that have worked before, which allows the scheduler to adapt as backlog and on-chain typologies drift and addresses the closed-loop retraining gap identified in Section VI. To keep this safe, the agent is constrained to a regulator-approved band of thresholds and its choices are logged for audit.

H. Alerting, Feedback and Retraining Loop

Escalated cases are routed to human investigators, whose disposition (confirmed suspicious, cleared, or escalated to a Suspicious Activity Report) is captured both as a labelled training example and as the reward signal R_t in Eqs. (9)–(10). Periodic incremental retraining folds confirmed dispositions back into the GNN encoder, which closes the loop between model output and ground truth in a way that most reviewed studies, trained once on a static snapshot, do not support [1]–[9].

Two cautions apply. Dispositions arrive with a delay, so the retraining job must tolerate labels that describe events from weeks earlier. And because analysts only review what the model escalates, the feedback is biased toward what the model already finds suspicious; a small random audit sample of non-escalated cases is therefore recommended to keep the training data honest. Where several institutions or exchanges take part, this layer can adopt the vertically and horizontally partitioned federated training pattern described by Kadhe et al. [19].

I. Deployment and Scalability Considerations

The layers are intended to be deployed as independent services connected through a message queue, so that ingestion can keep pace with the chain even if scoring momentarily lags. Scoring can be scaled horizontally by partitioning the graph on address ranges or clusters, with a small overlap region to preserve cross-partition neighbourhoods. Model versions are stored alongside the graph snapshot they were trained on, so that any alert can be reproduced later, which is a practical necessity for audit.

J. An Illustrative Scenario

To show how the layers cooperate, consider a hypothetical case. A set of addresses receives a large sum from a source previously tagged by an exchange as high-risk. Within a short time, the funds are split into many smaller transfers (structuring), sent through a chain of intermediate addresses that each forward most of the value (a peeling pattern), and finally re-converge on an address that interacts with a mixing service. The example is invented for illustration and does not come from any dataset.

When the blocks containing these transfers arrive, L1 records the addresses, edges and tags. L2 updates only the neighbourhoods around the new edges and refreshes their embeddings. L3 notices that several nodes on the path have high attention on the risk-tagged source and on the later mixer-tagged node, and the scores rise above the current threshold τ chosen by L5. L4 matches the subgraph to the structuring, peeling and mixing templates and writes a rationale naming the source, the hop sequence and the mixer. An investigator then confirms the case and files a report, which L6 records as a positive label and a reward. If a similar pattern later appears with slightly different timing, the retrained encoder should recognise it a little more readily. This is the closed loop that the offline studies in Section III do not describe.

K. Cost and Latency Considerations

A full re-embedding of the graph after every block costs time proportional to the number of edges in the window, which grows without bound as the chain does. The incremental approach instead limits work to the k-hop neighbourhood of the changed nodes, where k is the number of attention layers. The cost per block is therefore roughly proportional to the number of new edges multiplied by the typical neighbourhood size within k hops, and the sampling of hub neighbours keeps the worst case

bounded. We deliberately avoid quoting target numbers here, because latency depends on the chain's block interval, the hardware and the window size. Section VIII proposes how to measure it, and a service-level objective should be set per chain only after those measurements are available.

L. Model Lifecycle and Governance

A model that influences compliance decisions needs the same discipline as any regulated software. We suggest that every model version be registered together with its training window, feature definitions, hyperparameters and evaluation report, and that promotion from a shadow deployment to live use require a comparison against the incumbent on a recent validation window. In shadow mode the new model scores traffic without raising alerts, which lets the team observe score distributions and the agreement with the current model before anyone depends on it.

Monitoring after deployment should cover both inputs and outputs. Shifts in the distribution of input features, in the mean risk score or in the share of escalated cases are early signs of drift and should trigger a review, not silent retraining. Rollback must be possible at any time. These procedures are ordinary in machine-learning operations, but they are rarely written into the architecture of research prototypes, and they are what a supervisor is likely to ask about first.

M. Interfaces Between Layers

Finally, a reference architecture should say what crosses each boundary, because that is where integration problems arise. Table VIII lists the main messages exchanged between layers. Keeping these as explicit, versioned schemas allows each layer to be replaced (for instance, swapping the scoring model) without disturbing the rest of the pipeline.

TABLE VIII
MAIN MESSAGES EXCHANGED BETWEEN LAYERS

From → To	Message	Key fields
L1 → L2	Graph delta	Node/edge ids, types, timestamps, features
L2 → L3	Refreshed neighbourhood	Affected ids, updated embeddings
L3 → L5	Scored transactions	Id, score, model version
L5 → L4	Escalated cases	Id, score, threshold used
L4 → L6	Alert package	Typology, subgraph, rationale
L6 → L3	Training feedback	Disposition, label, reward, delay

N. Reliability and Security of the Platform Itself

A monitoring platform becomes a target in its own right. Its inputs come from public chains that anyone can write to, so an adversary can try to poison the system by creating large numbers of cheap transactions meant to shift feature statistics or to flood the queue with decoys. The layers therefore include basic safeguards: rate and size limits on any single entity's influence on the graph, outlier checks on incoming features, and alerts when the volume of escalated cases changes abruptly. The threshold agent is further bounded so that no sequence of inputs can push it to a value outside the approved band.

Reliability matters just as much. If a node of the ingestion layer falls behind or restarts, the graph must be rebuilt consistently from the last confirmed block and not from an arbitrary point. We recommend idempotent processing of blocks, so that replaying a block never produces duplicate edges, and handling of chain reorganizations by marking recent blocks as provisional until they are sufficiently confirmed. These are standard distributed-systems concerns, yet they directly affect whether the scores an analyst sees can be trusted.

O. Implementation Roadmap

We suggest building the platform in stages, so that value is demonstrated early and risk is kept low. The first stage implements L1 and L2 for a single chain and verifies that the incremental graph matches a graph rebuilt from scratch. The second adds the scoring layer L3 with a simple GCN baseline, followed by the directed heterogeneous model, and evaluates both offline. The third adds explanation (L4) and a shadow-mode alert queue. The fourth introduces the threshold agent (L5) first in simulation, using logged analyst decisions, and only later in live operation. The last stage connects the feedback loop (L6) and, if several institutions participate, the federated training of [19]. Each stage yields something that can be tested on its own, which keeps the project manageable for a small team.

VIII. PROPOSED EVALUATION METHODOLOGY

Because ChainSentry-GNN is presented as a reference architecture, we propose that its evaluation proceed in two stages. First, the graph-attention scoring layer of Eqs. (5)–(7) should be validated on the Elliptic and Elliptic2 benchmarks [1], [2] to confirm relational-modelling performance against established baselines under standard offline metrics. Second, the full pipeline (ingestion, incremental update, scoring and explanation) should be evaluated on a replayed, time-ordered stream of historical on-chain data, to capture system behaviour that offline benchmarks cannot: end-to-end graph-update latency, sustained inference throughput and explanation turnaround time. The scoring layer should additionally be tested on at least one non-Bitcoin chain to examine the cross-chain generalization identified as a gap in Section VI.

A. Baselines and Protocol

We suggest comparing against a random forest on the tabular features, GCN, GAT and GraphSAGE under matched settings [11], and where feasible the ensemble of [14]. All models should be trained on the earliest time steps and tested on the latest, with hyperparameters tuned on an intermediate validation window only. Results should be averaged over several random seeds and reported with confidence intervals, since single-run comparisons on a dataset with so few illicit labels can be misleading.

B. Metrics

Table IX lists the proposed metrics. They deliberately extend beyond classification accuracy to capture system readiness. Detection quality is reported through illicit-class F1 and PR-AUC; operational burden through the false-positive rate and the alert-to-SAR conversion rate; and system behaviour through update latency, throughput, explanation fidelity and cross-chain performance drop.

TABLE IX
PROPOSED EVALUATION METRICS FOR CHAINSENTRY-GNN

Metric	Definition	Target
Illicit-class F1 / PR-AUC	Precision–recall balance on the minority illicit class	Maximize
False-positive rate	Share of legitimate addresses/transactions incorrectly escalated	Minimize
Alert-to-SAR conversion	Proportion of escalated alerts confirmed as reportable	Maximize
Graph-update latency	Time to incorporate a new on-chain block into the live graph	Minimize
Inference throughput	Transactions scored per second at steady state	Maximize
Explanation fidelity	Agreement between surfaced rationale and model decision boundary	Maximize
Cross-chain generalization	Performance drop when applied to an unseen chain or wallet cluster	Minimize

C. Ablation and Sensitivity Studies

To establish which parts of the design earn their cost, we propose ablations that remove one component at a time: replacing the incremental update by periodic full rebuilds, replacing directed heterogeneous attention by a plain GCN, removing the typology layer, and replacing the adaptive threshold by a fixed one. A sensitivity analysis of the reward coefficients α, β and γ in Eq. (9), and of the escalation threshold τ selected through Eq. (10), is also proposed, mirroring the operator-facing trade-off analyses used in adjacent adaptive fraud-detection research [20]. The aim is to show how different settings shift the balance among detection sensitivity, false-positive load on investigators and graph-update latency under realistic transaction volumes.

D. Robustness and Reproducibility

Adversarial drift can be probed by simulating evasive behaviour on the replayed stream, for example by inserting additional intermediate hops or splitting transfers more finely, and measuring how quickly the score and the threshold agent recover. For reproducibility, we recommend releasing the preprocessing scripts, split definitions and random seeds, and documenting the compute budget given to each baseline, since that information is often missing from published comparisons.

E. Threats to Validity

Three threats should be acknowledged in advance. Internal validity may suffer from label leakage if features aggregated over a neighbourhood use information from time steps later than the transaction being scored, so aggregates must be computed from past data only. External validity is limited by the chains and periods used, and conclusions drawn from replayed history may not hold on a live feed whose burstiness differs. Construct validity concerns the proxies we use for investigator cost and regulatory risk in Eq. (9): they are estimates and should be calibrated with the institution that will use the platform.

F. Proposed Experimental Plan

In practice we envisage the evaluation as four sequential experiments, each of which has to succeed before the next is worth running. Experiment one reproduces published Elliptic baselines to check that our data handling and split protocol agree with the literature. Experiment two evaluates the directed heterogeneous scoring layer against those baselines, including the ablations described above. Experiment three runs the replayed stream through the complete pipeline and records update latency, throughput and explanation turnaround. Experiment four applies the scoring layer, with its feature definitions unchanged, to a second chain and quantifies the performance drop.

Reporting should follow the same checklist for every experiment: the exact split, number of seeds, hardware, tuning budget and the full set of metrics in Table IX, not only the favourable ones. Negative or mixed results are informative here, since they indicate which layer of the architecture is the real bottleneck.

G. Expected Outcomes and Success Criteria

Because this is a reference architecture, we frame success criteria as hypotheses to be tested and not as promises. We expect directed heterogeneous attention to match or improve on the plain GCN baseline on illicit-class F1 and PR-AUC; we expect the incremental update to produce substantially lower per-block latency than periodic full rebuilds while giving embeddings that remain close to

the rebuilt ones; and we expect the adaptive threshold to reduce analyst load at comparable recall relative to a fixed cut-off. Any of these hypotheses may fail, and a clear failure, reported honestly, would still be a useful finding for the community.

IX. REGULATORY, ETHICAL AND PRIVACY CONSIDERATIONS

A detection platform operates in a regulated and sensitive setting, so technical performance is only part of what must be right. First, a risk score is not proof of wrongdoing. Escalation should lead to human review, and nothing in the proposed design automates a decision to freeze funds or file a report without an analyst. Wrongly flagging a legitimate user has real costs, so false-positive monitoring and a route for review are treated as part of the system and not as optional extras.

Second, pseudonymous does not mean free of privacy concerns. Linking addresses into clusters can come close to identifying individuals, particularly once combined with exchange data. The platform should therefore follow data-minimization principles, restrict access to cluster metadata and keep audit logs of who looked at what. Third, fairness deserves attention: if some regions, exchanges or asset types are over-represented in the labelled data, the model may over-flag them, and error rates should be reported per segment where the data allow.

Finally, any natural-language explanation generator must stay grounded in the evidence that the model actually used. A fluent but unfaithful rationale is arguably worse than none, since it can lend false confidence to a weak alert. For this reason the explanation layer is built around typology templates populated from the graph, rather than free text generation alone.

A last point concerns dual use. Techniques that expose laundering patterns can, in principle, be studied by those who wish to avoid detection. We take the view that the benefit of transparent, reviewable detection methods outweighs this risk, particularly since the typologies described here are already widely discussed in the public literature, but responsible release of any implementation should still consider what is published and in how much detail.

X. LIMITATIONS AND FUTURE SCOPE

This paper has limits of its own. It is a review combined with a proposed design, and the architecture has not yet been implemented or benchmarked, so we make no claims about its measured accuracy or speed. The review covers eighteen studies chosen for their relevance to GNN-based blockchain AML and does not claim to be exhaustive; the field is moving quickly, and newer work may already address some of the gaps identified here. Some of the literature we rely on is available as preprints that may not have completed peer review.

Future work falls into four directions. The first is empirical validation of the architecture on multi-chain data, using the replay protocol of Section VIII. The second is robustness testing under adversarial on-chain restructuring. The third is a closer connection between federated cross-institution training and real-time inference, so that exchanges can share learning without sharing customer records. The fourth is the extension of the explanation layer to cross-chain typologies such as chain-hopping, which require linking activity across ledgers.

XI. CONCLUSION

This paper reviewed eighteen studies on graph neural networks for blockchain anti-money laundering, spanning foundational Elliptic-benchmark models, attention-based and heterogeneous architectures, temporal and ensemble methods, explainability and privacy-preserving techniques, and early real-time system designs. The analysis shows that GNNs have become the dominant and consistently effective way to capture coordinated on-chain laundering behaviour. It also shows that the field has concentrated heavily on offline detection accuracy over a single blockchain and a single historical period, with comparatively little attention to incremental graph maintenance, inference latency, cross-chain generalization and architecture-native, investigator-facing explanation.

To address these gaps, the proposed ChainSentry-GNN architecture combines chain-agnostic ledger ingestion, incremental graph updating, directed heterogeneous graph-attention scoring (Eqs. 5–7), reinforcement-learning-driven adaptive thresholding (Eqs. 9–10), typology-aligned explanation and a closed feedback loop with human investigators in a single platform design. By treating update latency, throughput and explanation fidelity as first-class evaluation criteria alongside detection accuracy, the proposed direction aims to help move blockchain AML research from static, benchmark-optimized models toward deployable, auditable monitoring platforms.

REFERENCES

[1] M. Weber, G. Domeniconi, J. Chen, D. K. I. Weidele, C. Bellei, T. Robinson, and C. E. Leiserson, “Anti-Money Laundering in Bitcoin: Experimenting with Graph Convolutional Networks for Financial Forensics,” in Proc. KDD Workshop on Anomaly Detection in Finance, 2019.

[2] C. Bellei, M. Xu, R. Phillips, T. Robinson, M. Weber, T. Kaler, C. E. Leiserson, Arvind, and J. Chen, “The Shape of Money Laundering: Subgraph Representation Learning on the Blockchain with the

Elliptic2 Dataset," in Proc. KDD Workshop on Machine Learning in Finance, 2024, arXiv:2404.19109.

[3] I. Alarab, S. Prakoonwit, and M. I. Nacer, "Competence of Graph Convolutional Networks for Anti-Money Laundering in Bitcoin Blockchain," in Proc. 5th Int. Conf. Machine Learning Technologies (ICMLT), Beijing, China, 2020, pp. 23–27.

[4] I. Alarab and S. Prakoonwit, "Graph-Based LSTM for Anti-Money Laundering: Experimenting Temporal Graph Convolutional Network with Bitcoin Data," Neural Process. Lett., vol. 54, pp. 1–19, 2022.

[5] W. W. Lo, M. Sarhan, S. Layeghy, and M. Portmann, "Inspection-L: A Self-Supervised GNN-Based Money Laundering Detection System for Bitcoin," arXiv:2203.10465, 2022.

[6] D. Cheng, Y. Ye, S. Xiang, Z. Ma, Y. Zhang, and C. Jiang, "Anti-Money Laundering by Group-Aware Deep Graph Learning," IEEE Trans. Knowl. Data Eng., vol. 35, no. 12, pp. 12444–12457, 2023.

[7] X. Li, Y. Li, X. Mo, H. Xiao, Y. Shen, and L. Chen, "DIGA: Guided Diffusion Model for Graph Recovery in Anti-Money Laundering," in Proc. 29th ACM SIGKDD Conf. Knowledge Discovery and Data Mining, 2023, pp. 4404–4413.

[8] O. M. Olaniyi, I. S. Aroh, O. Henry, O. C. Metibemu, and O. I. Akinola, "Graph Neural Networks for Multi-Layered Financial Crime Network Detection: An Explainable AI Framework for Anti-Money Laundering," J. Eng. Res. Rep., vol. 28, no. 2, pp. 18–36, 2026.

[9] C.-H. Poon, J. Kwok, C. Chow, and J.-H. Choi, "LineMVGNN: Anti-Money Laundering with Line-Graph-Assisted Multi-View Graph Neural Networks," AI, vol. 6, no. 4, art. 69, 2025.

[10] F. Johannessen and M. Jullum, "Finding Money Launderers Using Heterogeneous Graph Neural Networks," arXiv:2307.13499, 2023.

[11] S. Marasi and S. Ferretti, "Anti-Money Laundering in Cryptocurrencies Through Graph Neural Networks: A Comparative Study," in Proc. 2024 IEEE 21st Consumer Communications & Networking Conf. (CCNC), Las Vegas, NV, USA, 2024.

[12] B. Deprez, T. Vanderschueren, B. Baesens, T. Verdonck, and W. Verbeke, "Network Analytics for Anti-Money Laundering: A Systematic Literature Review and Experimental Evaluation," arXiv:2405.19383, 2024.

[13] S. Chen, Y. Liu, Q. Zhang, Z. Shao, and Z. Wang, "Multi-Distance Spatial-Temporal Graph Neural Network for Anomaly Detection in Blockchain Transactions," Adv. Intell. Syst., vol. 7, no. 8, art. 2400898, 2025.

[14] M. Z. Haider, T. Noreen, and M. Salman, "Towards Quantum-Ready Blockchain Fraud Detection via Ensemble Graph Neural Networks," arXiv:2509.23101, 2025.

[15] G. M. Ntuala, Q. Xia, H. Xia, A. Badjie, P. Mukala, E. E. Da Silva Tavares, J. Gao, and C. C. Ukwuoma, "A Dual-Layer GNN with Economic Penalty Mechanisms for Blockchain Fraud Detection," Expert Syst. Appl., 2025.

[16] P. Li, Y. Xie, X. Xu, J. Zhou, and Q. Xuan, "Phishing Fraud Detection on Ethereum Using Graph Neural Network," arXiv:2204.08194, 2022.

[17] K. Khanvilkar and K. Kommuru, "Regulatory Graphs and GenAI for Real-Time Transaction Monitoring and Compliance Explanation in Banking," arXiv:2506.01093, 2025.

[18] O. Lawal, A. Okolie, and C. Obunadike, "An Explainable Graph Neural Network Framework for Anti-Money Laundering in Cryptocurrency Transactions Using the Elliptic Dataset," Int. J. Network Security & Its Applications, vol. 17, no. 5/6, 2025.

[19] S. R. Kadhe, H. Ludwig, N. Baracaldo, A. King, Y. Zhou, K. Houck, A. Rawat, M. Purcell, N. Holohan, M. Takeuchi, R. Kawahara, N. Drucker, H. Shaul, E. Kushnir, and O. Soceanu, "Privacy-Preserving Federated Learning over Vertically and Horizontally Partitioned Data for Financial Anomaly Detection," arXiv:2310.19304, 2023.

[20] Y. Cui, X. Han, J. Chen, X. Zhang, J. Yang, and X. Zhang, "FraudGNN-RL: A Graph Neural Network With Reinforcement Learning for Adaptive Financial Fraud Detection," IEEE Open J. Comput. Soc., vol. 6, pp. 426–437, 2025.