

Temporal-Context Edge Enrichment for Financial Crime Detection

Danae Natalie Savvidi

Delft University of Technology

Temporal-Context Edge Enrichment for Financial Crime Detection

by

Danae Natalie Savvidi

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday September 24, 2026.

Student number: 5457769
Project duration: March 4, 2026 – September 24, 2026
Thesis committee: Prof. dr. Arie van Deursen, TU Delft, chair
Dr. Kubilay Atasu, TU Delft, daily supervisor
Dr. Asterios Katsifodimos, TU Delft, external member

Cover: Generated using ChatGPT Images 2.0 (OpenAI)
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.

Preface

This thesis concludes my Master of Science in Data Science and Artificial Intelligence Technology at Delft University of Technology. The months I worked on the thesis became a highlight of the six years I have spent at TU Delft, which perhaps should not come as a surprise.

That I ended up in research is no coincidence. Science runs in my family. Most of them are *theoretical* physicists, my father surprised everyone by becoming an *experimental* physicist, and my mother is a chemist. I followed in his footsteps, not into physics but in surprising everyone, and chose computer science. The curiosity, however, I certainly inherited. The idea that a difficult problem is something to enjoy rather than avoid is what carried me through this project.

For my thesis I wanted to work on something with an impact in the real world, and helping to detect financial crime turned out to be an excellent fit. It was also a genuine challenge. Reliable detection is not an easy feat, with criminal techniques that keep evolving, an enormous amount of data to search through, and very little of that data labelled well enough to learn from. Working on this project has helped me grow both professionally and personally.

I was fortunate to have a great deal of help along the way. I would like to sincerely thank my daily supervisor Dr. Kubilay Atasu for pointing me towards this topic in the first place. I am grateful to have had the opportunity to work with him, and would like to thank him and my advisor Çağrı Bilgi for their guidance throughout this project, and in particular for the constant push to do better and to keep looking for a more novel way to solve the underlying problem. I would also like to thank Prof. dr. Arie van Deursen for his detailed feedback on this work, as well as for chairing my thesis committee, and Dr. Asterios Katsifodimos for the time and interest he devoted to this thesis as an external committee member. Finally, I want to thank my incredibly supportive family and friends, who were always there for me. I could not have done this without them.

I hope you enjoy reading this thesis.

Danae Natalie Savvidi
Delft, September 2026

Abstract

Financial crime detection requires identifying rare illicit activity within large transaction networks, where suspicious behaviour can depend on both relationships between accounts and patterns in their transaction histories. Existing graph-based methods capture structural information, yet typically represent temporal behaviour only through transaction attributes or shared account representations. This thesis investigates whether representations from pretrained time-series foundation models (TSFMs) can provide richer context for graph-based financial crime detection. We introduce Temporal-Context Edge Enrichment (TCEE), which uses Chronos-2, a pretrained TSFM, to encode account transaction histories and enables each transaction to condition the retrieval of account-history context on its own features. Unlike approaches that assign all transactions of an account the same representation, TCEE constructs a transaction-specific representation conditioned on the transaction itself. These edge representations can be classified directly or given to a graph neural network (GNN) for additional structural information. Experiments on six AMLWorld anti-money laundering datasets show that TCEE without a GNN backbone improves PR-AUC over the strongest evaluated baseline on five datasets by 10.8% on average, while reaching its best validation PR-AUC $6.0\times$ faster end-to-end on average across all six datasets, including the one-time cost of Chronos-2 encoding. Adding a GNN backbone improves PR-AUC by a further 4.8% over TCEE without the backbone on the four datasets trained to convergence, and 2.4% on average across all six. On a real Ethereum phishing graph, Chronos-PNA improves PR-AUC by 14.5% over the strongest evaluated baseline, while full TCEE-PNA improves it by 17.8%, showing that Chronos-2 representations can also transfer beyond anti-money laundering when combined with graph learning.

Contents

Preface	i
Abstract	ii
Nomenclature	v
1 Introduction	1
2 Background	3
2.1 Graph Neural Networks	3
2.1.1 Graph Representation	3
2.1.2 Message Passing	4
2.1.3 Message Passing on Directed Multigraphs	5
2.1.4 Aggregation Functions	5
2.1.5 Node and Edge Classification	7
2.2 Time-Series Representation Learning	7
2.2.1 Multivariate Time Series	7
2.2.2 Irregularly Sampled Time Series	8
2.2.3 Representation Learning for Time Series	8
2.3 Time-Series Foundation Models	8
2.3.1 Foundation Models	9
2.3.2 Pretraining on Time Series	9
2.3.3 Patch-Based Temporal Modelling	9
2.3.4 Chronos-2	10
2.4 Attention Mechanisms	10
2.4.1 Motivation for Attention	10
2.4.2 Query, Key and Value	11
2.4.3 Scaled Dot-Product Attention	11
2.4.4 Self-Attention and Cross-Attention	12
2.5 Evaluation under Class Imbalance	12
3 Scientific Paper	13
4 Further Discussion	22
4.1 Interpreting the Main Results	22
4.2 Efficiency and Applicability	23
4.3 Interpretability and Actionability	23
4.3.1 Feature Attention	23
4.3.2 Neighbourhood Attention	24
4.4 Generalisability and Transfer Beyond AML	24
4.5 Societal Impact and Ethics	24
4.6 Limitations	24
4.7 Future Research	25
4.8 Concluding Remarks	25
References	26
A Dataset Statistics and Experimental Details	28
A.1 Dataset Statistics	28
A.2 Laundering Typologies	29
A.3 Pair Structure	29
A.4 Neighbourhood Window Size	31

A.5	Experimental Details	31
B	Additional Ablations	33
B.1	Contribution of Chronos and Pretraining	33
B.1.1	Chronos Contribution	33
B.1.2	Pretraining Contribution	34
B.2	Information Sources	34
B.2.1	Effect of Time and Counterparty Information	34
B.2.2	Representing Time and Counterparty	34
B.2.3	Account Views of the Transaction	35
B.3	Pair Context Design	35
B.4	Neighbourhood Window	36
B.5	Architectural Sensitivity	37
B.5.1	Projection Width	37
B.5.2	Attention Width	37
B.5.3	Edge-Conditioned Retrieval	38
B.5.4	Parameter Sharing	39
C	Attention Analysis	40
C.1	Measurement Protocol	40
C.2	Attention Selectivity	40
C.2.1	Attention Concentration	40
C.2.2	Account-Level and Transaction-Level Variation	41
C.2.3	Feature-wise Attention	42
C.3	Attention and Feature Importance	43
C.3.1	What the Model Needs	43
C.3.2	Feature Attention Deletion	43
C.3.3	Neighbourhood Attention Deletion	44
C.4	Attention and Pattern Identification	45
C.4.1	Identifying Pattern-Related Transactions	45

Nomenclature

Abbreviations

Abbreviation	Definition
1-WL	One-dimensional Weisfeiler–Lehman test, the graph isomorphism test that upper-bounds message-passing expressive power
AML	Anti-money laundering
AMLWorld	Synthetic anti-money laundering benchmark suite, six datasets over two illicit ratios and three sizes
ETH	Ethereum, the blockchain of the phishing detection dataset
F1	Harmonic mean of precision and recall, reported for the illicit minority class
GIN	Graph Isomorphism Network
GINE	Graph Isomorphism Network extended with edge features
GNN	Graph neural network
GPU	Graphics processing unit
HI / LI	Higher and lower illicit ratio regimes of the AMLWorld benchmark
LN	Layer normalisation, which rescales a vector to zero mean and unit variance across its components
MLP	Multilayer perceptron
PNA	Principal Neighbourhood Aggregation
PR-AUC	Area under the precision-recall curve
TCEE	Temporal-Context Edge Enrichment, the method introduced in the article
TSFM	Time-series foundation model

Symbols

Symbol	Definition
$G = (V, E)$	Transaction multigraph with accounts V and transactions E
$e = (u, v, k)$	Transaction from sender u to receiver v with features $\mathbf{z}_e$, where k indexes parallel edges. The article writes it $e = (u, v, t, \mathbf{z}_e)$
$y_e, \hat{y}_e$	True label of a transaction and its predicted illicit probability
$\mathbf{x}_v$	Input feature vector of node v
$\mathcal{N}(v)$	Neighbourhood of node v , split by direction into $\mathcal{N}_{\text{in}}(v)$ and $\mathcal{N}_{\text{out}}(v)$
$\mathbf{h}_v^{(l)}$	Representation of node v after l message-passing layers, with L layers in total
AGG	Permutation invariant aggregation function of a message-passing layer
f_θ, U_θ	Learned message and update functions of a message-passing layer
α, s, λ	PNA's aggregators, its degree scalars, and the scaling factor they are built from
$\mathbf{e}_e^{(l)}$	Representation of edge e after l layers, starting from $\mathbf{e}_e^{(0)} = \mathbf{z}_e$
$\mathbf{h}_{uv}$	Pooled representation of the K_{uv} parallel edges from u to v
$\mathbf{X}$	Multivariate time series of F variables over T time steps, in $\mathbb{R}^{F \times T}$
$x^{(f)}$	The f -th row of $\mathbf{X}$, one variable over the whole sequence, called a series
$\parallel$	Concatenation of vectors
$\mathbf{E}$	Learned representation, or embedding, of a time series
Δt_i	Time gap between two consecutive observations
p	Patch length, the number of time steps a patch covers

F	Number of feature series per account (six on AMLWorld)
P_a	Number of patches in account a 's encoded history
$\mathbf{E}_a$	Frozen embedding tensor of account a , in $\mathbb{R}^{F \times P_a \times 768}$
$\mathbf{q}, \mathbf{K}, \mathbf{V}$	Query vector, key matrix and value matrix of an attention operation
d_k, d_v	Dimensionality of the queries and keys, and of the values
n_q	Number of queries of an attention operation
β_i	Attention weight assigned to the i -th key
$\mathbf{c}$	Output of an attention operation, a weighted combination of its values
$\mathbf{ctx}$	Temporal context vector an edge retrieves from an account's embeddings
d_{proj}	Width of the learned projection of the frozen embeddings
d_{attn}	Width of the attention space in TCEE
w	Half-width of the neighbourhood attention window, which reads $2w$ transactions per account
M_{uv}	Multiset of the transactions between accounts u and v , in either direction
L_a	History length of account a , the number of transactions incident to it

1

Introduction

Financial transactions are an essential part of everyday life, but the same digital infrastructure that enables them can also be exploited to move the proceeds of crime. Money laundering, fraud and other illicit activities are carried out through sequences of transactions. The scale of the problem is significant. Money laundering alone is estimated at 2 to 5% of global GDP [1], and in 2025 complaints to the FBI's Internet Crime Complaint Center reported \$11.4 billion of losses in the United States alone to crimes involving cryptocurrency [2]. Finding illicit activity is not simply a matter of identifying unusual transactions. Money launderers often deliberately structure their activity to appear ordinary and blend into the much larger population of legitimate transactions. Banks are nonetheless required to monitor their transactions and report suspicious activity [3].

Financial transactions naturally form a graph. Each account can be represented as a node, and each transaction between two accounts can be represented as a directed edge. Since the same pair of accounts can exchange money repeatedly, this forms a transaction multigraph. Figure 1.1 shows the eight laundering shapes that the AMLWorld benchmark generates in a transaction network [4]. What has to be flagged depends on the crime. In anti-money laundering the task is to determine whether an individual transaction is illicit or legitimate. This is known as edge classification. In contrast, other forms of financial crime detection, such as phishing detection on the Ethereum blockchain [5], require identifying suspicious accounts, which makes the task node classification.

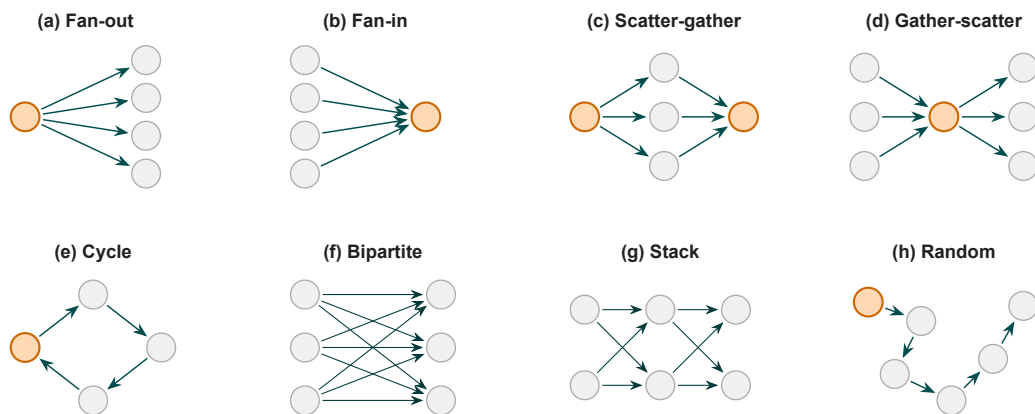


Figure 1.1: The eight shapes of laundering that the AMLWorld benchmark generates [4]. Circles are accounts and arrows are transfers. Orange marks the account the money starts from, ends at, or passes through. The bipartite and stack shapes have no such account, because they move money between two sets of accounts rather than through one.

Classifying an edge or a node is rarely done in isolation. Looking at the neighbouring nodes and edges provides context for the decision. Modern graph-based machine learning models, particularly graph neural networks (GNNs), have been designed to exploit this type of relational structure. Rather than

treating every transaction as an independent observation, they allow information to be propagated through the transaction network so that a prediction can take neighbouring accounts and transactions into account. However, relationships within the network are not the only source of information.

Financial behaviour is temporal. Legitimate accounts often have recurring patterns, such as regular salaries and expenses. Two transactions with similar amounts and positions in the graph may have different meanings depending on what happened before and after them. Time-based features have been shown to distinguish suspicious from non-suspicious accounts on real banking data [6].

The two kinds of information, relational and temporal, are difficult to combine. Graph-based models are effective at representing relational information, but treat time as just another transaction attribute [7, 8, 9]. Temporal graph models are built to additionally model time, and a common approach is to compress an account's history into a single account-level representation [10, 5]. That representation does not depend on which transaction is being classified. For an edge classification task this is restrictive, because different transactions may need to consider different aspects of an account's history, such as the amounts involved, the timing, or the counterparties.

We propose a way to combine the relational and temporal information in a more flexible manner. Recent time-series foundation models (TSFMs) [11] are pretrained on large collections of time-series data and learn general patterns in temporal sequences. They are trained to forecast, but the representations they build have been shown to transfer usefully to domains they never saw during pretraining [12]. We use the transaction history of each account as input to the TSFM, which returns one representation for each aspect of that history instead of a single summary of the account. This also avoids the need to train a temporal model from scratch using the relatively scarce illicit samples. What remains is to determine how an account's resulting representations should be used to classify individual transactions.

The key idea explored in this thesis is that in edge classification, each transaction should have its own temporal context rather than relying on a shared account-level representation. We propose *Temporal-Context Edge Enrichment* (TCEE), which builds a separate representation for every transaction from its own features, its accounts' histories and its local context. On the six AMLWorld datasets TCEE beats the strongest evaluated baselines on five of them, raising the area under the precision-recall curve by 10.8% on average across those five. Averaged over all six datasets, it reaches its best score $6.0\times$ faster than each dataset's strongest baseline, with the one-time cost of encoding the account histories included. We additionally test on a real Ethereum graph, where feeding the account history representations to a GNN beats every evaluated baseline, this time at detecting phishing accounts.

This thesis addresses four research questions. The first three concern anti-money laundering transaction classification, while the fourth examines transfer to phishing detection.

RQ1 *How effective is TCEE for anti-money laundering transaction classification?*

RQ2 *What does TCEE cost to train?*

RQ3 *How much do frozen TSFM account history representations contribute?*

RQ4 *How well do TSFM representations transfer beyond anti-money laundering to phishing detection?*

The remainder of this thesis is structured as follows. Chapter 2 presents the background the scientific article assumes, covering graph neural networks and graph-based classification, what it means to learn a representation of a time series, the time-series foundation models that can produce such a representation, the attention operation TCEE is built from, and how results are scored when illicit transactions are rare. Chapter 3 is the scientific article *Temporal-Context Edge Enrichment for Financial Crime Detection*, which presents the method, the experimental results and discussion in publication format. Chapter 4 interprets the main findings, discusses the limitations of the approach, outlines future research, and considers the societal impact and ethics of deploying TCEE. Appendices A to C contain dataset statistics, experimental details for reproducibility, further ablations and experiments, and an analysis of the attention weights.

2

Background

This chapter covers the material needed to follow the article in Chapter 3. Section 2.1 covers graph neural networks, the models that learn from a transaction network, the two aggregation functions this thesis uses, and the node and edge classification tasks they are applied to. Section 2.2 covers time series, what makes them different from unordered data, and what it means to learn a representation of one. Section 2.3 narrows this to pretrained time-series models and to Chronos-2, the encoder this thesis uses to represent account histories. Section 2.4 covers attention, the main mechanism behind the proposed approach. Section 2.5 closes with how results are scored, since illicit transactions are rare enough that the choice of metric decides what a result means.

2.1. Graph Neural Networks

Many real-world datasets cannot be represented naturally as simple tables. Different objects can form relationships with one another. For example, in a social network, users can be friends with one another, and in a transaction network, accounts can send money to one another. These relationships are naturally represented as a graph, where the objects are nodes and the relationships are edges. Graph neural networks (GNNs) are a class of machine learning models designed to operate on such graph-structured data.

The central idea behind a GNN is that the representation of a node should depend not only on its own features, but also on the features of the nodes it is connected to. This is typically achieved through a mechanism called *message passing*, in which nodes exchange information with their neighbours and update their representations. By repeatedly performing this operation, information can propagate through increasingly large parts of the graph.

2.1.1. Graph Representation

A graph can be represented as $G = (V, E)$ where V is the set of nodes and E is the set of edges. A node $v \in V$ represents an entity in the data, and an edge $(u, v) \in E$ represents a relationship between two entities. For example, in a social network a node could represent a user and an edge could represent a friendship between two users. Each node $v \in V$ can have a *node feature* vector $\mathbf{x}_v$ which describes the entity, such as the age of a user. Edges can also have an *edge feature* vector $\mathbf{z}_{uv}$ that describes the relationship between nodes u and v , such as how close a friendship is. The *neighbourhood* $\mathcal{N}(v)$ is the set of nodes joined to v by an edge, and it is the only part of the graph that v can see directly. From these inputs a graph model produces a *node representation* $\mathbf{h}_v$, a vector that summarises the node together with its surroundings. The article in Chapter 3 keeps this symbol for accounts, writing $\mathbf{h}_a^{(l)}$ for the state of account a after l layers and $\mathbf{h}_a$ for a summary of the account that no message passing produced.

Graphs can be directed or undirected. In a directed graph, edges are ordered pairs, so (u, v) is not the same as (v, u) . Graphs can also be simple or multigraphs. In a simple graph there is at most one edge between any two nodes, while a multigraph allows several parallel edges between the same pair.

The graphs studied in this thesis are *directed multigraphs*. Specifically, a transaction has a sender and a receiver, so an edge carries a direction, and multiple transactions can occur between the same pair of accounts over time. Each edge is therefore written as $e = (u, v, k) \in E$, where the index k distinguishes parallel edges, and it carries its own feature vector $\mathbf{z}_e$ rather than one shared vector $\mathbf{z}_{uv}$ per pair. Because transactions are timestamped, the index k can be replaced by the time of the transaction, and the article in Chapter 3 writes the same edge as $e = (u, v, t, \mathbf{z}_e)$. Direction also splits the neighbourhood into the accounts that paid v and the accounts that v paid,

$$\mathcal{N}_{\text{in}}(v) = \{u \mid (u, v, k) \in E\}, \quad \mathcal{N}_{\text{out}}(v) = \{u \mid (v, u, k) \in E\}, \quad (2.1)$$

with $\mathcal{N}(v) = \mathcal{N}_{\text{in}}(v) \cup \mathcal{N}_{\text{out}}(v)$ when the direction is ignored.

2.1.2. Message Passing

The main mechanism used by many GNNs is message passing [13]. The intuition is that each node receives information from its neighbours and uses this information to update its own representation. Figure 2.1 illustrates the mechanism. Let $\mathbf{h}_v^{(l)}$ denote the representation of node v at layer l , with the initial representation given by its input features, $\mathbf{h}_v^{(0)} = \mathbf{x}_v$.

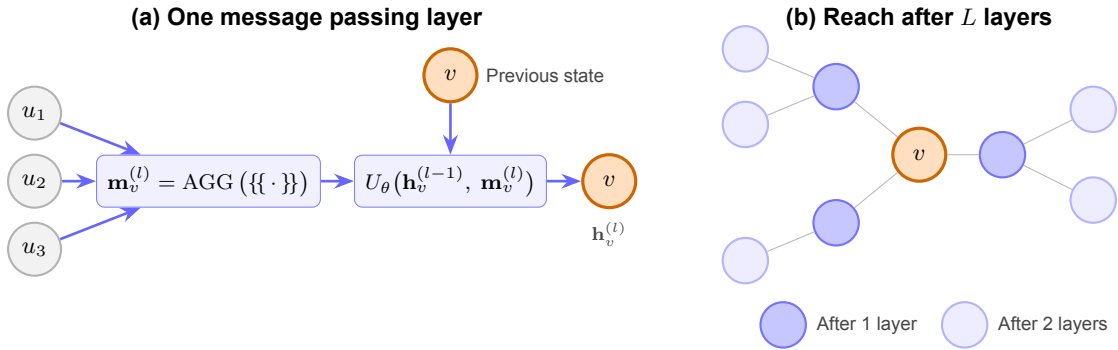


Figure 2.1: Message passing. (a) A node v aggregates its neighbours' representations and updates its own from the result. (b) Stacking layers widens what v has seen, so the number of layers sets how far the model looks.

At layer l , node v receives one message from each neighbouring node $u \in \mathcal{N}(v)$. A general message-passing layer can be written as:

$$\begin{aligned} \mathbf{m}_v^{(l)} &= \text{AGG} \left(\left\{ \left\{ f_{\theta}^{(l)}(\mathbf{h}_v^{(l-1)}, \mathbf{h}_u^{(l-1)}, \mathbf{z}_{uv}) : u \in \mathcal{N}(v) \right\} \right\} \right), \\ \mathbf{h}_v^{(l)} &= U_{\theta}^{(l)}(\mathbf{h}_v^{(l-1)}, \mathbf{m}_v^{(l)}), \end{aligned} \quad (2.2)$$

where $\mathbf{z}_{uv}$ denotes the features of edge (u, v) . The message function $f_{\theta}^{(l)}$ determines what information node u sends to node v . It may use the representation of the receiving node, the representation of the neighbouring node, and the features of the connecting edge. The aggregation function AGG then combines all messages received by v into a single vector $\mathbf{m}_v^{(l)}$. The messages form a *multiset*, written $\{\{\cdot\}\}$, rather than a set, because two neighbours can send identical messages and both have to count [14]. Finally, the update function $U_{\theta}^{(l)}$ combines this aggregated message with the previous representation of node v to produce its new representation. Figure 2.1a follows these steps at a single node. The functions $f_{\theta}^{(l)}$ and $U_{\theta}^{(l)}$ are learned during training and can be implemented, for example, with multilayer perceptrons.

When neighbours of a node do not have a natural ordering, the aggregation function must be permutation invariant. In other words, reordering the messages must not change the aggregated result. Common choices include summation, mean, and maximum. The choice of aggregation function can affect the *expressive power* of the GNN, meaning which graph structures it is able to tell apart [14, 15], but the best choice varies depending on the dataset and task.

Message passing also determines how far information can travel through the graph. After one layer, the representation of node v can depend on its immediate neighbours. After two layers, it can depend

on nodes that are at most two hops away, and after L layers it can incorporate information from an L -hop neighbourhood, as Figure 2.1b shows. Increasing the number of layers therefore increases the receptive field of each node. However, using many layers can also lead to over-smoothing, where representations of different nodes become increasingly similar [16].

2.1.3. Message Passing on Directed Multigraphs

Equation (2.2) is written for a simple undirected graph, where each neighbour is connected to v by a single edge and sends one message. A transaction graph differs from this because it is a directed multigraph where multiple transactions may connect the same pair of accounts, and the transaction has a direction. A model therefore needs to distinguish both repeated interactions between the same accounts and money flowing into and out of an account.

Multi-GNN [7] addresses these properties by combining bi-directional message passing [17] with multi-graph port numbering and Ego-IDs [18]. Bi-directional message passing allows a node to receive information from both its incoming and outgoing neighbours, while port numbers distinguish parallel edges and edges belonging to different neighbours. This makes it possible for the model to represent directed multigraph structure that a standard GNN may otherwise treat as equivalent.

MEGA-GNN [8] keeps bi-directional message passing and Ego-IDs and replaces port numbering with multi-edge aggregation built into message passing itself. The key idea is to aggregate parallel edges between each pair of accounts first, and then aggregate the resulting messages across distinct neighbours. Every edge carries its own representation $\mathbf{e}_e^{(l)}$, starting from its features, $\mathbf{e}_e^{(0)} = \mathbf{z}_e$. For a pair in which u pays v , the K_{uv} parallel edges are first combined into a single pair representation,

$$\mathbf{h}_{uv}^{(l-1)} = \text{EdgeAgg} \left(\left\{ \left\{ \mathbf{e}_e^{(l-1)} : e = (u, v, k) \in E, k \in [1, \dots, K_{uv}] \right\} \right\} \right), \quad (2.3)$$

after which message passing runs over neighbours rather than over edges, with one message per distinct neighbour built from the sender and the pooled pair representation,

$$\mathbf{m}_{\text{in},v}^{(l)} = \text{AGG} \left(\left\{ \left\{ f_{\theta}^{(l)}(\mathbf{h}_u^{(l-1)}, \mathbf{h}_{uv}^{(l-1)}) : u \in \mathcal{N}_{\text{in}}(v) \right\} \right\} \right). \quad (2.4)$$

The receiver enters only through the update. The outgoing direction is handled by a mirrored construction with its own edge representations, aggregation, and message function, producing $\mathbf{m}_{\text{out},v}^{(l)}$. The update then combines the two aggregates,

$$\mathbf{h}_v^{(l)} = U_{\theta}^{(l)}(\mathbf{h}_v^{(l-1)}, \mathbf{m}_{\text{in},v}^{(l)}, \mathbf{m}_{\text{out},v}^{(l)}). \quad (2.5)$$

Merging the parallel edges does not replace them. Each transaction keeps its own representation, refreshed from the representation of the account that sent it, its own previous state, and the pooled representation of the edges it runs parallel to,

$$\mathbf{e}_e^{(l)} = g_{\theta}^{(l)}(\mathbf{h}_u^{(l-1)}, \mathbf{e}_e^{(l-1)}, \mathbf{h}_{uv}^{(l-1)}), \quad e = (u, v, k). \quad (2.6)$$

The reversed edges are updated by their own function in the same way. Both stages are repeated at every layer, so the pooled representation is rebuilt each time from the current state of the individual transactions. Two transactions between the same pair of accounts therefore stay distinguishable, which matters because the label being predicted belongs to one transaction rather than to the pair. How these representations are turned into a prediction is described in Section 2.1.5.

2.1.4. Aggregation Functions

The aggregation function is an important component of message passing because it determines how information from a node's neighbours is combined. Different choices capture different properties and can affect the expressive power of the resulting model. This section describes the two aggregation methods used in this thesis, the Graph Isomorphism Network (GIN) and Principal Neighbourhood Aggregation (PNA).

Graph Isomorphism Network

The Graph Isomorphism Network (GIN) was introduced by Xu et al. [14] to study the expressive power of message-passing graph neural networks. Xu et al. showed that the expressive power of standard message-passing GNNs is upper-bounded by the one-dimensional Weisfeiler–Lehman (1-WL) test. GIN is significant because it is exactly as powerful as the 1-WL test, achieving this theoretical maximum for message-passing GNNs through a relatively simple architecture based on sum aggregation followed by a multilayer perceptron (MLP).

The GIN update rule is given by:

$$\mathbf{h}_v^{(l)} = \text{MLP}^{(l)} \left((1 + \epsilon^{(l)}) \cdot \mathbf{h}_v^{(l-1)} + \sum_{u \in \mathcal{N}(v)} \mathbf{h}_u^{(l-1)} \right), \quad (2.7)$$

where $\epsilon^{(l)}$ controls the relative contribution of the node's own representation and can either be learned or fixed.

The original GIN formulation does not explicitly incorporate edge features, and treats edges as binary connections. GINE extends the approach by adding the edge features to the neighbour representation before aggregation [19],

$$\mathbf{h}_v^{(l)} = \text{MLP}^{(l)} \left((1 + \epsilon^{(l)}) \cdot \mathbf{h}_v^{(l-1)} + \sum_{e=(u,v,k) \in E} \text{ReLU}(\mathbf{h}_u^{(l-1)} + \mathbf{z}_e) \right), \quad (2.8)$$

where the sum runs over the edges pointing into v rather than over its neighbours, so that each parallel transaction contributes its own term. This is the form used in this thesis. A further point is that the GIN result above concerns countable features, whereas transaction data are largely continuous. Corso et al. extend the analysis to continuous features and show that discriminating between multisets of size n drawn from the reals requires at least n aggregators, so no single aggregation function suffices however expressive the network around it is [15]. This motivates Principal Neighbourhood Aggregation (PNA), which uses several aggregation functions to capture different properties of a neighbourhood.

Principal Neighbourhood Aggregation

Principal Neighbourhood Aggregation (PNA) was proposed by Corso et al. [15] to capture more information about a node's neighbourhood than a single aggregation function can provide. Instead of summarising incoming messages using only one statistic, PNA applies several aggregators to the same multiset of messages. In the formulation used here, it applies four aggregators α to the multiset M of incoming messages, rescales each result by three degree scalers s , and concatenates the twelve outputs,

$$\begin{aligned} \text{PNA}(M) &= \parallel_{\alpha, s} s \alpha(M), \quad \alpha \in \{\text{mean}, \text{std}, \text{max}, \text{min}\}, \\ s &\in \{1, \lambda, \lambda^{-1}\}, \quad \lambda = \log(1 + |M|) / \mu, \end{aligned} \quad (2.9)$$

where $\parallel$ denotes concatenation and μ , the mean of $\log(1 + \deg)$ over the training set, normalises λ to average one.

Edge features enter through the message function that forms the multiset. Each message is computed from both endpoints and the edge between them, and the aggregated result is passed through a second transform together with the node's own representation,

$$\begin{aligned} \mathbf{m}_e^{(l)} &= \text{MLP}_{\text{pre}}^{(l)}(\mathbf{h}_v^{(l-1)} \parallel \mathbf{h}_u^{(l-1)} \parallel \mathbf{z}_e), \quad e = (u, v, k) \in E, \\ \mathbf{h}_v^{(l)} &= \text{MLP}_{\text{post}}^{(l)}(\mathbf{h}_v^{(l-1)} \parallel \text{PNA}(M_v^{(l)})), \end{aligned} \quad (2.10)$$

where $M_v^{(l)}$ collects the messages $\mathbf{m}_e^{(l)}$ of the edges pointing into v .

The motivation for using several aggregators is particularly clear for continuous and heterogeneous data. For example, consider two accounts whose neighbours have the same average transaction amount. One account might receive transactions whose amounts are all similar, while the other receives

a mixture of very small and very large transactions. Their means could therefore be similar even though their neighbourhoods are quite different. The standard deviation, minimum, and maximum provide additional information that can distinguish these cases. This makes PNA well suited to graphs where both the features and the number of neighbours can vary substantially [15], such as transaction networks.

2.1.5. Node and Edge Classification

One important application of GNNs is node classification. In this task, the objective is to predict a label for each node. For example, in a social network, the task could be to predict whether a user is likely to be interested in a particular product based on their connections and attributes. The GNN first produces a representation $\mathbf{h}_v^{(L)}$ for each node v after L layers of message passing. A classifier can then be applied to this representation to predict the label $\hat{y}_v$ of node v .

Another important application is edge classification, where the goal is to predict a label for each edge in the graph. For example, in a transaction network, the task could be to predict whether a transaction is fraudulent or not. After message passing, each node has a representation $\mathbf{h}_v^{(L)}$. For an edge $e = (u, v, k)$ connecting nodes u and v , these representations can be combined to form an edge representation $\mathbf{e}_e = \rho(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)}, \mathbf{z}_e)$, where ρ could be concatenation, addition, or another function. The resulting representation can then be passed to a classifier to predict the label $\hat{y}_e$ of edge e .

The edge features $\mathbf{z}_e$ already influence message passing, since the messages carry the features of the edges along which they are passed. However, the edge does not maintain its own hidden representation. As a result, the edge being classified is represented only indirectly through $\mathbf{h}_u^{(L)}$ and $\mathbf{h}_v^{(L)}$, which combine information from all edges involving the two nodes. A solution is to maintain a separate representation for each edge and update it alongside the node representations:

$$\mathbf{e}_e^{(l)} = R_\theta^{(l)} \left(\mathbf{e}_e^{(l-1)}, \mathbf{h}_u^{(l-1)}, \mathbf{h}_v^{(l-1)} \right), \quad (2.11)$$

where $\mathbf{e}_e^{(0)}$ is set from the original edge features $\mathbf{z}_e$. The updated edge representation is then used in the next message-passing layer, so that $\mathbf{z}_e$ in Equations (2.8) and (2.10) is replaced by $\mathbf{e}_e^{(l-1)}$, and the final representation $\mathbf{e}_e^{(L)}$ replaces $\mathbf{z}_e$ in the classifier, which therefore reads $\rho(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)}, \mathbf{e}_e^{(L)})$. The edge representation is combined with the representations of the two accounts rather than used on its own. Models built for anti-money laundering follow this design [8, 9, 7].

2.2. Time-Series Representation Learning

Time-series representation learning concerns the problem of representing data that evolve over time in a way that preserves information about their temporal behaviour. In the context of financial transactions, an individual transaction may be informative on its own, but the sequence of transactions associated with an account can reveal behavioural patterns that are not visible from isolated observations.

A time series is a sequence of observations recorded over time. Let $x_{1:T} = (x_1, x_2, \dots, x_T)$ denote a sequence containing T observations, where each x_t represents the observation at time t . An important property of time series is the presence of *temporal dependencies*. A temporal dependency occurs when the behaviour of an observation is related to observations that occurred before or after it.

2.2.1. Multivariate Time Series

Sequences of observations can be univariate or multivariate. A *univariate* time series consists of a single variable measured over time, while a *multivariate* time series consists of multiple variables captured simultaneously over time. These variables are not necessarily independent. Their values may be related to one another and their relationships may change over time.

This thesis focuses on multivariate time series, which can be represented as a matrix

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_T] \in \mathbb{R}^{F \times T}, \quad (2.12)$$

where F is the number of variables and T is the number of time steps. Each column $\mathbf{x}_t \in \mathbb{R}^F$ represents

the values of all variables at time step t . Each row $x^{(f)} \in \mathbb{R}^T$ is one variable followed over the whole sequence, and this thesis calls it a *series*, the term the article in Chapter 3 uses.

The transaction history of an account is naturally multivariate. For example, an individual transaction can contain information such as the transaction amount, whether money was sent or received, the currency involved, the payment type, and other transaction attributes.

2.2.2. Irregularly Sampled Time Series

In regularly sampled time series, observations arrive at a fixed interval such as every second, minute, or day. However, not all time series are observed at regular time intervals. Transaction data generally do not have this property. One transaction may occur a few seconds after the previous transaction, while the next may not occur for several hours or days.

Let t_i denote the timestamp of the i -th observation in a time series. The time elapsed between two consecutive observations can be expressed as

$$\Delta t_i = t_i - t_{i-1}. \quad (2.13)$$

This quantity is referred to as the *inter-event* time or time gap between observations. In this thesis, the first observation of a sequence is given a gap of zero since it has no predecessor.

The time gap can itself contain useful information about the process being observed. For example, two accounts could have identical sequences of transaction types and amounts, but very different temporal behaviour if one account performs these transactions within a few minutes while the other performs them over several weeks. Ignoring the timestamps would make these two sequences appear more similar than they actually are.

For transaction histories, this means that the temporal structure is not determined only by the order of transactions. The amount of time separating consecutive transactions can also contribute to the representation of an account's behaviour.

2.2.3. Representation Learning for Time Series

The goal of representation learning is to automatically transform complex input data into a more compact representation that captures useful information about the underlying data. For a time series, this can be expressed abstractly as

$$\mathbf{X} \mapsto \mathbf{E}, \quad (2.14)$$

where $\mathbf{X}$ is the input time series of Equation (2.12) and $\mathbf{E}$ is the learned representation, also called an *embedding*.

An embedding is a numerical representation of the input that places information about the original data into a form that can be used by subsequent machine-learning models. The representation does not necessarily preserve every detail of the original sequence, but can capture characteristics that are useful for the intended task. In the temporal setting, these characteristics can include patterns in the sequence, relationships between variables, changes in behaviour, and dependencies between observations.

In this context, an account's transaction history can be understood as a multivariate, potentially irregularly sampled time series. Temporal representation learning provides a way of converting this sequence of events into learned temporal features that summarise the behavioural information contained in the history. The specific model used in this thesis to obtain these representations is introduced in Section 2.3.4.

2.3. Time-Series Foundation Models

The approach proposed in this thesis uses a pretrained time-series foundation model, Chronos-2, to extract information from transaction histories before performing the downstream financial crime detection task. This section introduces the concepts needed to understand this approach. First, the general idea of foundation models is described, followed by what changes when the pretraining data are time series. The concept of patch-based temporal modelling is then introduced, before discussing Chronos-2 specifically.

2.3.1. Foundation Models

A *foundation model* is a machine learning model that is pretrained on a large and diverse collection of data and can subsequently be adapted or applied to a range of downstream tasks. Rather than learning a representation from scratch for every individual task, the model first learns general patterns from large-scale pretraining data. These patterns can then be reused when the model is applied to a particular problem.

The central idea can therefore be viewed as a two-stage process. During *pretraining*, the model learns representations that capture useful patterns in the input data. During *downstream application*, these pretrained representations are used for a specific task, either by adapting the model through further training, namely *fine-tuning*, or by keeping the model *frozen* (not updating the model weights) and using it directly. This approach is particularly useful when the dataset available for the downstream task is relatively small or has high class imbalance, since the model does not have to learn all relevant patterns from that dataset alone.

2.3.2. Pretraining on Time Series

Time-series foundation models extend the general concept of foundation models to sequential numerical data, the time series X of Equation (2.12).

A *time-series foundation model*, or TSFM, is pretrained on many time-series datasets, usually with a forecasting objective, after which it can be applied to new time-series problems *zero-shot*, meaning with no training on them at all [20, 21]. The family is growing quickly and includes Moirai [22], TimesFM [23], MOMENT [24] and Chronos-2 [11].

A practical consideration in the setting of financial transaction data is that most TSFMs, including Chronos-2, are designed under the assumption that the input data are regularly sampled in time [21, 11]. In financial transaction data, however, this assumption does not strictly hold, since transactions are event-based and occur irregularly over time. To address this mismatch, the spacing between events has to be exposed to the model as part of its input, for example through the time gap Δt_i of Equation (2.13).

2.3.3. Patch-Based Temporal Modelling

Long time series can contain hundreds or thousands of individual observations. Processing every observation as a separate input element can be computationally expensive and can also make it more difficult for a model to capture patterns occurring over a longer time interval. One approach to addressing this problem is *patching* [25], shown in Figure 2.2.

Instead of treating a time series as a sequence of individual observations, patch-based modelling divides the time series into segments of fixed length p , called *patches*, so a series of T steps yields $\lceil T/p \rceil$ of them. Each patch represents a short period of the original time series rather than a single point in time. The model can transform each patch into a vector representation, often referred to as an *embedding*. When the series is multivariate, each of the F variables is patched on its own, so the model returns one embedding for every variable and every patch rather than one for every time step.

The important intuition is that an embedding is not simply a copy of the original observations. It is a vector produced by the model that encodes information about the corresponding part of the sequence. Because the representations are produced by a temporal model, they can also contain information about the surrounding temporal context.

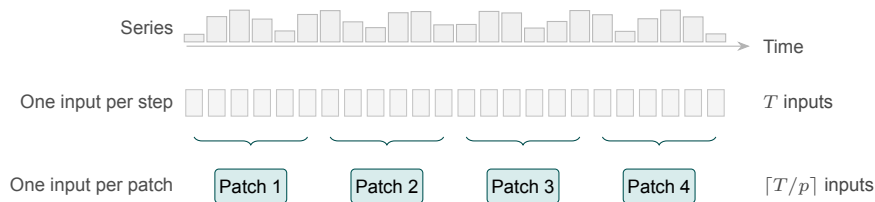


Figure 2.2: Patching a time series. Treating every one of the T time steps as a separate input gives each one a single scalar to carry. Patching instead cuts the series into contiguous segments of length p and treats each segment as one input, which shortens the sequence by a factor p .

2.3.4. Chronos-2

Chronos-2 [11] is a time-series foundation model developed by Amazon for general-purpose, zero-shot forecasting. It is an encoder-only model with approximately 120 million parameters and is designed to operate on both univariate and multivariate time-series problems within a single architecture.

An important property of Chronos-2 for this work is its ability to process *multivariate* time series. A multivariate time series contains several variables observed at each time step of the sequence. Rather than treating these variables as completely independent sequences, Chronos-2 can model relationships between them. The model achieves this using a *group attention* mechanism, which allows information to be shared between related series or variables [11].

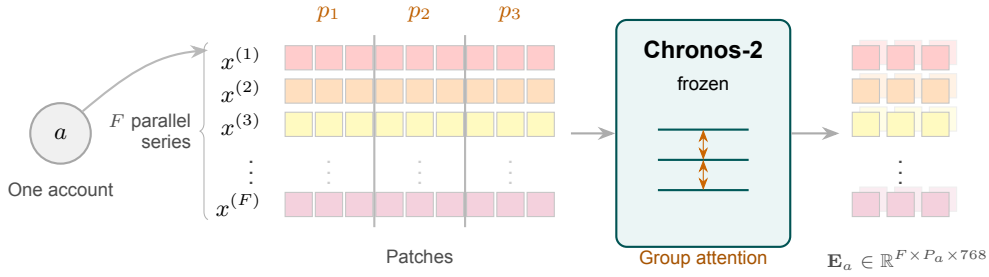


Figure 2.3: What a frozen multivariate encoder consumes and what it returns. An account's history is F parallel series, one per transaction attribute, which Chronos-2 patches and processes jointly, returning one 768 dimensional embedding per series and per patch.

Chronos-2 can be used as a feature extractor rather than being trained to perform the final financial crime classification task. In this thesis it is also kept *frozen*, so its pretrained parameters are never updated on financial crime data. The transaction history of each account is passed through the model once, and the resulting embeddings are what the downstream classifier consumes. Its output is a collection of embeddings rather than a single vector. For an account a whose F series each split into P_a patches, the encoder returns

$$E_a \in \mathbb{R}^{F \times P_a \times 768}, \quad (2.15)$$

one 768 dimensional embedding for every series and every patch.

Whether a generic pretrained encoder carries useful signal in a domain it never saw during pretraining is an empirical question, and the evidence so far is encouraging. StarEmbed [12] benchmarks time-series foundation models on astronomical variable-star light curves. Chronos embeddings reach the best reported clustering and out-of-distribution detection scores on that data, despite the model being pretrained on regularly sampled, non-astronomical time series. On supervised classification no model tested beats the long-established domain-specific baseline, so the transfer is not uniform across tasks. The representations nonetheless transfer usefully to an unseen domain, which motivates using a pretrained encoder as a temporal feature extractor rather than purely for forecasting.

2.4. Attention Mechanisms

Attention is a general neural-network mechanism that determines which parts of an input representation are most relevant for producing an output. Since different pieces of information may contribute unequally to a prediction, attention assigns them different learned weights based on their relevance. It is widely used in modern time-series foundation models [21].

2.4.1. Motivation for Attention

Consider a model that has to produce one output from n inputs, such as the steps of a sequence or the elements of a set, and let $\mathbf{H}$ be the matrix whose n rows are their representations $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_n$. The simplest way to reduce them to a single vector is to average them, but the result is then the same whatever the model is being asked. Attention instead assigns a weight β_i to each representation $\mathbf{h}_i$ and computes

$$\mathbf{c} = \sum_{i=1}^n \beta_i \mathbf{h}_i, \quad \beta_i \geq 0, \quad \sum_{i=1}^n \beta_i = 1. \quad (2.16)$$

The weights are learned from the data and depend on the information currently being processed, so the same $\mathbf{H}$ can be reduced to different vectors for different questions. It is this dependence, rather than the weighting itself, that the method of this thesis relies on.

2.4.2. Query, Key and Value

The attention mechanism formalises the process of determining relevance using three components: a *query* ($\mathbf{Q}$), a *key* ($\mathbf{K}$) and a *value* ($\mathbf{V}$). This formulation was introduced in the Transformer architecture by Vaswani et al. [26].

Let

$$\mathbf{Q} \in \mathbb{R}^{n_q \times d_k}, \quad \mathbf{K} \in \mathbb{R}^{n \times d_k}, \quad \mathbf{V} \in \mathbb{R}^{n \times d_v}, \quad (2.17)$$

where n_q is the number of queries, n is the number of keys and values, d_k is the dimensionality of the queries and keys, and d_v is the dimensionality of the values.

The three components have different roles. The *query* $\mathbf{Q}$ represents the information for which the model is currently looking. It can be interpreted as the “question” being asked by the model. The *key* $\mathbf{K}$ represents the information used to determine whether a particular input is relevant to the query. Keys can therefore be viewed as identifiers against which the query is compared. The *value* $\mathbf{V}$ contains the actual information that is passed to the output once the relevance of each key has been determined. In short, the query is compared with each key to determine its relevance. The resulting relevance scores are converted into weights, and the corresponding values are then combined according to these weights.

Typically $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are obtained through learned linear projections of underlying representations. For the input representation matrix $\mathbf{H}$, these projections can be written as

$$\mathbf{Q} = \mathbf{H}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{H}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{H}\mathbf{W}_V, \quad (2.18)$$

where $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$ are trainable parameter matrices [26]. This means the model learns how the original representations should be transformed into queries, keys, and values during training.

2.4.3. Scaled Dot-Product Attention

Scaled dot-product attention is defined as in [26]

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right) \mathbf{V}. \quad (2.19)$$

The equation performs three main operations, calculating similarities, converting them into weights, and combining values. First, the product $\mathbf{Q}\mathbf{K}^\top$ calculates the similarity between each query and key. The scores are then scaled by $\frac{1}{\sqrt{d_k}}$, where d_k is the key dimensionality. This scaling prevents the scores from becoming excessively large when the key vectors have many dimensions [26]. Lastly, the scaled scores are passed through the softmax function. The method in this thesis always attends with one query at a time, so $n_q = 1$ and the query is a single vector $\mathbf{q}$. It also gives the keys and the values the same width, $d_k = d_v = d$. The article in Chapter 3 therefore writes every attention it uses in the single-query form

$$\text{Attn}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{q}\mathbf{K}^\top}{\sqrt{d}}\right) \mathbf{V}, \quad \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d}. \quad (2.20)$$

For a particular query, the attention weight assigned to key $\mathbf{k}_i$ is

$$\beta_i = \frac{\exp\left(s_i/\sqrt{d}\right)}{\sum_{j=1}^n \exp\left(s_j/\sqrt{d}\right)} \quad (2.21)$$

where $s_i = \mathbf{q}\mathbf{k}_i^\top$ is the similarity score between the query $\mathbf{q}$ and key $\mathbf{k}_i$. These β values are commonly referred to as *attention weights*. A high β_i means that the corresponding value $\mathbf{v}_i$ has a relatively large contribution to the output for the given query.

2.4.4. Self-Attention and Cross-Attention

The attention mechanism has two important variants, *self-attention* and *cross-attention*. The distinction is based on the origin of the queries, keys, and values.

In *self-attention* the query, the keys and the values all come from the same representation. Although Q , K , and V originate from the same input representation, they have different learned projections and therefore serve different roles. This is what Chronos-2 does both along the patches of a series and across the parallel series of an account. Chronos-2 is an encoder, so every patch attends to every other patch of the encoded history, bidirectionally.

In *cross-attention* the query comes from one representation while the keys and values come from another. For example, in this thesis the query is built from a single transaction, while the keys and values come from temporal representations of its sender and receiver accounts. Thus, although the keys and values remain fixed for a given account, their contribution to the output changes with the query. In this sense, the same temporal representation can be “read” differently depending on the transaction providing the query.

A related approach is to use a fixed learned query to attend to an input-dependent representation. This is used for attention-based pooling in the *Set Transformer*, where Pooling by Multihead Attention (PMA) uses learnable seed (query) vectors to attend to the elements of an input set [27]. The authors use a single seed vector in most cases, resulting in a single pooled representation. In this setting, the query is shared across inputs, while the keys and values depend on the particular input. The article pools each account’s feature embeddings into a single vector in exactly this way, with a learned query.

2.5. Evaluation under Class Imbalance

Financial crime data are severely imbalanced, since illicit transactions constitute a small fraction of all transactions, as little as 0.051% on the sparsest of the AMLWorld datasets [4]. Accuracy is therefore uninformative in this setting, because a model that labels every transaction as legitimate can achieve a high accuracy despite failing to identify illicit activity. Results are consequently reported using the F_1 score of the illicit minority class and the area under the precision-recall curve (PR-AUC).

For the binary classification setting considered here, the illicit class is treated as the positive class. A true positive (TP) is an illicit transaction correctly classified as illicit, a false positive (FP) is a legitimate transaction incorrectly classified as illicit, and a false negative (FN) is an illicit transaction incorrectly classified as legitimate. On a node classification task the same definitions apply with accounts as the unit and the illicit accounts as the positive class. Precision and recall are then defined as

$$P = \frac{TP}{TP + FP}, \quad R = \frac{TP}{TP + FN}, \quad (2.22)$$

and the F_1 score is their harmonic mean:

$$F_1 = \frac{2PR}{P + R} = \frac{2TP}{2TP + FP + FN}. \quad (2.23)$$

Both quantities depend on a decision threshold, since a model outputs a score rather than a label. Sweeping the threshold traces out a precision-recall curve, and PR-AUC summarises that curve in a single number. Because the test set is finite, the curve is observed at a finite set of points, one for each distinct score, so the area is computed as a sum rather than as an integral. Writing P_n and R_n for the precision and recall at the n -th threshold, with thresholds taken in decreasing order so that recall grows with n and $R_0 = 0$,

$$\text{PR-AUC} = \sum_n (R_n - R_{n-1}) P_n, \quad (2.24)$$

which weights the precision reached at each threshold by the recall it gains over the previous one. This quantity is also known as average precision.

3

Scientific Paper

The following pages contain the scientific article that forms the core of this thesis,

Temporal-Context Edge Enrichment for Financial Crime Detection

Danae Natalie Savvidi, Çağrı Bilgi, Kubilay Atasu and Arie van Deursen.

The article was submitted to the ACM International Conference on AI in Finance (ICAIF 2026) and is included here in the two column format of that venue. Every concept it assumes is developed in Chapter 2. Additional results and explanations can be found in the Appendix.

Temporal-Context Edge Enrichment for Financial Crime Detection

Danae Natalie Savvidi
Delft University of Technology
Delft, The Netherlands

Çağrı Bilgi
Delft University of Technology
Delft, The Netherlands

Kubilay Atasu
Delft University of Technology
Delft, The Netherlands

Arie van Deursen
Delft University of Technology
Delft, The Netherlands

Abstract

Financial crime detection requires identifying illicit activity within large transaction networks, where suspicious behaviour is rare and unfolds over time. Existing graph-based methods capture relationships among accounts but typically underuse the temporal behaviour encoded in account transaction histories. We introduce *Temporal-Context Edge Enrichment* (TCEE), a framework that addresses this gap by summarising each account’s transaction history with a frozen time-series foundation model (TSFM), and enriching each transaction’s representation with the relevant temporal context from these summaries. These enriched representations can be scored directly for edge classification, or fed into a graph neural network (GNN) as edge features to capture broader structural information. Across six AMLWorld datasets, TCEE improves PR-AUC over the strongest evaluated baseline on five by an average of 10.8% while training 6× faster end-to-end. Feeding TCEE’s representations into a GNN provides a further 4.8% PR-AUC improvement on the four smaller datasets. On an Ethereum phishing dataset, we show that TSFM-generated account representations fed into a GNN are also effective for node classification improving PR-AUC by 14.5% over the strongest baseline. As TSFMs continue to advance, our results suggest they can strengthen financial crime detection by capturing temporal signals that complement graph-based approaches.

Keywords

financial crime detection, anti-money laundering, time-series foundation models, graph neural networks

1 Introduction

Modern crime is not only physical. Financial crime is carried out through digital transactions, from money laundering to defrauding account holders. Money laundering alone is estimated to account for 2 to 5% of global GDP each year [24]. Whatever the crime, illicit funds must move, and every movement is recorded as a transaction between two accounts, so financial crime unfolds on a transaction network. Such a network is naturally a *multigraph*, with accounts as nodes, transactions as directed edges, and parallel edges representing repeated transactions between account pairs.

Finding the few illicit transactions among millions of legitimate ones is the task of financial crime detection, a legal obligation for financial institutions [18]. GNNs and graph transformers designed for such multigraphs define the state of the art in banking and

blockchain settings alike [10, 11, 16], and recent work further extends them [3]. We call this family *structural models*, as they focus on the graph structure yet consume time as a plain edge attribute.

Despite advances, detection remains challenging. Illicit behaviour is vanishingly rare, as few as one transaction in two thousand [1], and the criminal patterns, sequences of coordinated transfers, evolve as criminals seek to evade detection [18]. What must be detected also varies with the crime. In phishing, such as on the Ethereum (ETH) blockchain, victims are tricked into paying an account the attacker controls, so the label is the account itself and detection is *node classification* [29]. Money laundering is different. The proceeds can travel through otherwise legitimate accounts, and what is illicit is each transfer that moves them, so anti-money laundering (AML) benchmarks pose the problem as *edge classification* [1]. Edge classification is generally harder, because whether a transaction is suspicious depends on context. The model must learn not only the characteristics of the accounts involved but also how they interact.

Among these challenges, we target the temporal signal. Accounts transact in repeating rhythms, salaries arrive and bills are paid on schedule, and on real bank data these rhythms separate launderers from legitimate customers [14]. Two obstacles stand in the way. Temporal patterns must be learned from scarce illicit labels, and the rhythms belong to accounts while the decision falls on a single transaction. The problem is therefore to enrich each transaction with temporal context drawn from its structural neighbourhood.

Prior work misses this requirement in different ways. The structural models reduce the temporal signal to a single timestamp attribute. Temporal graph models are built around time, yet each line of work reads it narrowly. Snapshot models see only the edges inside a time step [26], temporal graph networks compress an account’s past into one memory vector [20], and recency-weighted message passing reduces timing to one scalar per neighbour [12].

Time-series foundation models (TSFMs) such as Chronos-2 [2] offer a new option. Pretrained on large corpora, they supply rich temporal representations without needing any labels. A frozen TSFM takes each account’s history as one series per feature, the amount, currency and so on, and returns one embedding per series. Pooling this set into one vector per account, which we call *node-level fusion*, and using it as a node feature in a GNN works well for node-level tasks. On a real ETH phishing graph [5, 29], this model outperforms all evaluated baselines. However, it still trails

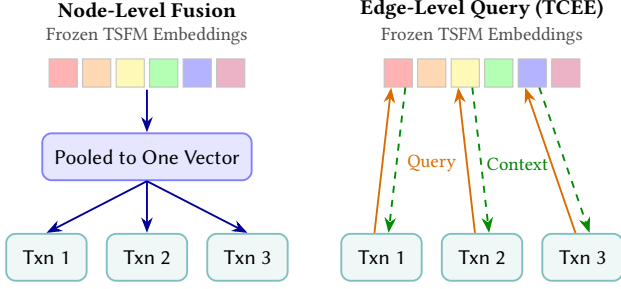


Figure 1: Node-level fusion (left) provides the same pooled vector to every transaction of the same account; TCEE (right) enables each transaction to query the context independently.

the strongest structural models on AML edge classification [1], because it hands every transaction of an account the same pooled vector, while edge-level tasks require telling them apart.

We overcome this limitation by delivering the temporal context directly to the edge. *Temporal-Context Edge Enrichment* (TCEE) lets each transaction form a query from its own features and attend over the frozen embeddings of its source and destination accounts (Figure 1), producing a rich edge representation. Two transactions by the same account can thus retrieve different contexts: e.g., one drawing on the amount history, and another on the timing. We further add two *extensions*, components that read the raw transaction features directly and produce further representation details the temporal context cannot provide. To our knowledge, TCEE is the first model to query a TSFM from a graph *edge*. The enriched edge representations built by TCEE can be scored directly by an MLP (TCEE-MLP) or handed to a GNN backbone that can consume edge features (TCEE-GNN).

We make the following main contributions:

- (1) **Models.** We build, to our knowledge, the first TSFM-based detectors of financial crime on transaction multigraphs, **Chronos-MLP** and **Chronos-GNN**, which use node-level fusion.
- (2) **Method.** We show that node-level fusion falls short on edge-level tasks and introduce **TCEE**, which enables each transaction to independently query its temporal context.
- (3) **Results.** On AMLWorld, TCEE-MLP with both extensions and no backbone improves PR-AUC, the area under the precision-recall curve, over each dataset’s strongest evaluated baseline on five of the six datasets, by 10.8% on average, and trains 6.0× faster end-to-end on average across all six. A GNN backbone adds a further 4.8% where trained to convergence. On the ETH phishing graph, TCEE improves PR-AUC by 17.8% over the strongest baseline. Together, these results make a case for using TSFMs in financial crime detection.

2 Background and Related Work

Problem. We work on a directed transaction multigraph $G = (V, E)$. Each node is an account and each edge a transaction $e = (u, v, t, z_e)$ from u to v at time t with raw features z_e . AML is edge classification, where a classifier $f : e \mapsto \hat{y}_e \in [0, 1]$ predicts the illicit label $y_e \in \{0, 1\}$ under severe class imbalance, with illicit

transactions as rare as 0.051% on AMLWorld LI-Medium [1]. Phishing detection is node classification, predicting one label per account.

Feature Engineering. Before GNNs, AML systems paired hand-engineered features with classifiers. Weber et al. [26] pair each transaction’s local attributes with one-hop neighbourhood statistics on the Elliptic Bitcoin graph, and Savage et al. [21] extract community structure and transaction dynamics per group for a supervised classifier. The Graph Feature Preprocessor [4] mines subgraph patterns in real time to feed gradient-boosted trees, matching or outperforming simple GNN baselines. Such features are fixed in advance, so patterns they do not express stay invisible to the classifier. GNNs instead learn features directly from the graph.

Structural GNNs. The strongest models compete on *structural* expressiveness, how faithfully the model interprets the graph. Multi-GNN [11] adds ego IDs, port numbering and reverse message passing to standard GNNs so they provably recognise directed subgraph patterns such as cycles. MEGA-GNN [3] aggregates in two stages. It first pools the parallel edges between an account pair, then combines the pooled messages across the account’s neighbours. Multi-FraudGT [16] uses the augmentations of Multi-GNN and performs message passing with a graph transformer whose attention is biased by edge features. On the phishing task, early detectors classify each account from a random-walk embedding of the transaction network [29]. DIAM [10], designed for cryptocurrency multigraphs, compresses each account’s transactions into one node embedding with a GRU. Multi-GNN, MEGA-GNN and Multi-FraudGT improve how graph structure is read, yet treat a transaction’s *time* as a plain edge attribute. DIAM’s GRU models time explicitly, but it is still trained end-to-end on the scarce illicit labels rather than pretrained. TCEE supplies what both lack, a pretrained temporal representation. It is *orthogonal* to the structural axis, changing only the edge representation consumed, so it composes with any of them.

Temporal Modelling. Several lines already model timing. Message passing can be re-weighted towards recent transactions [12]. Continuous-time dynamic graph networks attach a recurrent memory or a time encoding to each node [20], discrete-time ones evolve their weights over graph snapshots [26], and dedicated detectors exploit smurfing’s transfer timing [23]. The graph models learn their temporal representation from scratch, and each account is summarised by a single node embedding that all its transactions share. TCEE differs on both counts. It reads each account’s full history with a frozen pretrained model, a richer temporal signal, and it delivers that signal to the individual transaction, the unit AML classifies, rather than diluting it in node aggregation.

Frozen Time-Series Encoders. TSFMs are neural networks pretrained on large, diverse collections of time series, so they learn general temporal patterns and forecast new series with no task-specific training [2]. Most split each series into short, fixed-length, non-overlapping segments called patches, and treat them as input tokens [17]. Architectures range from encoder-only models such as Chronos-2 [2] and Moirai [28] to decoder-only forecasters such as TimesFM [8], and from univariate to multivariate models that jointly model related series. What matters for TCEE is a set of embeddings per account to attend over, not one pooled vector. Any TSFM with an encoder provides this, so the approach follows TSFM

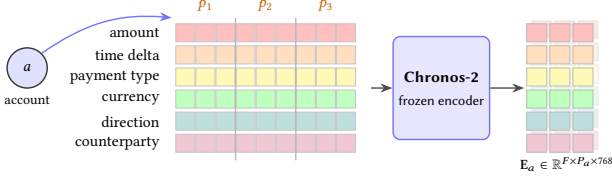


Figure 2: Chronos-2, frozen, reads each account’s history in patches of 16 transactions and emits one 768-d embedding per feature and patch.

advances rather than depending on one model. We use Chronos-2, a leading multivariate, encoder-only forecaster, kept frozen. Each account a has F parallel feature series, one per transaction feature (Section 3.1). Chronos-2 encodes the F series of an account jointly. Its group attention performs in-context learning, meaning that the representation of every feature series is conditioned on the account’s remaining $F - 1$ series without any weight updates. The encoder reads each in patches of 16 transactions, and returns the tensor $E_a \in \mathbb{R}^{F \times P_a \times 768}$ (Figure 2), one 768-dimensional embedding per series and patch, with P_a the account’s patch count. TCEE queries these embeddings with each transaction. Graph models that condition attention on edge features [9] or message-pass on the line graph [19] operate over representations learned inside the graph, and work that brings pretrained encoders to GNNs fuses them at the *node* level [22]. To our knowledge, TCEE is the first to query a frozen encoder from a graph *edge*, and to bring a TSFM to financial crime detection.

3 Method

This section covers the encoding of transaction histories as time series, node-level fusion, TCEE and the cost analysis. All attention below is single-head scaled dot-product attention [25], $\text{Attn}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{q}\mathbf{K}^\top / \sqrt{d})\mathbf{V}$ for a query $\mathbf{q}$, keys $\mathbf{K} \in \mathbb{R}^{n \times d}$ and values $\mathbf{V}$.

3.1 Transaction History as a Time Series

A transaction history is not the time series a TSFM expects, for two reasons. First, transactions are discrete events at irregular intervals, while time-series models often assume regular sampling. Second, each transaction involves two accounts, so a history is part of a graph, not a self-contained series. Our construction addresses both.

We build one sequence per account, containing every transaction it sends or receives, ordered by time. Each transaction is one step of the sequence. For AMLWorld each step carries $F=6$ features, so the history becomes six parallel series, one per feature (Figure 2). The *amount*, *payment type* and *currency* describe the transaction itself, and the *direction* flag marks it as incoming or outgoing for the account. The remaining two series close the two gaps above. The *time delta* $\Delta t_i = t_i - t_{i-1}$, in minutes, with t_i the time of the account’s i -th transaction and $\Delta t_1 = 0$, closes the first. Chronos-2 treats its input as equally spaced, so the delta is the simplest way to expose the spacing between steps. The *counterparty index* closes the second. It numbers the account’s counterparties in the order they first appear, so a returning partner keeps its number. Through this series the encoder sees the account’s one-hop neighbourhood,

so the representation it returns is not purely temporal. We ablate the counterparty index (Table 4) to measure how much of the signal is temporal. Categorical and boolean features become integers, as the official implementation accepts only real-valued series.

Before training, Chronos-2 encodes each account’s F series jointly into the embedding tensor $E_a \in \mathbb{R}^{F \times P_a \times 768}$ (Figure 2). We average E_a over its patches, $\bar{E}_a = \frac{1}{P_a} \sum_{p=1}^{P_a} E_a^{(p)} \in \mathbb{R}^{F \times 768}$, with $E_a^{(p)}$ the p -th patch, leaving one embedding per feature for each account. We ablate the averaging (Table 4).

3.2 Node-Level Fusion

A simple way to use the frozen embeddings is *node-level fusion*, pooling them into the same vector per account. Two such models measure what the frozen signal is worth on its own. Each model pools the account’s feature embeddings $\bar{E}_a$ into one vector $\mathbf{h}_a \in \mathbb{R}^{d_{\text{proj}}}$. By default we pool with attention. A linear map per feature projects $\bar{E}_a$ to $\tilde{E}_a \in \mathbb{R}^{F \times d_{\text{proj}}}$, and a learned query $\mathbf{q}_0$ attends over the projections,

$$\mathbf{h}_a = \mathbf{W}_o \text{Attn}(\mathbf{q}_0, \tilde{E}_a, \tilde{E}_a), \quad (1)$$

with $\mathbf{W}_o$ a learned output map. The attention lets each account weight its most informative series, and replacing its weights with a uniform mean gives a plain-mean variant we ablate (Table 4). Chronos-MLP classifies an edge from its two accounts,

$$\hat{y}_e = \text{MLP}([\mathbf{h}_u \parallel \mathbf{h}_v \parallel \mathbf{h}_u \odot \mathbf{h}_v \parallel |\mathbf{h}_u - \mathbf{h}_v| \parallel \mathbf{z}_e]), \quad (2)$$

with $\parallel$ denoting concatenation, $\odot$ the Hadamard product and $\mathbf{z}_e$ the edge’s raw features. The product and absolute difference are standard pair-matching features [6] that show the classifier how the two accounts align and differ. Chronos-GNN instead uses the same $\mathbf{h}_a$ as account a ’s initial node feature in the GNN backbone (Section 3.3.4).

3.3 TCEE: Temporal-Context Edge Enrichment

TCEE lets each transaction attend over the frozen embeddings of its two accounts, $\bar{E}_u$ and $\bar{E}_v$, with a query built from the transaction’s own features. Figure 3a shows the architecture.

3.3.1 The Edge-Conditioned Query.

Keys and Values. For each account a we project its averaged feature embeddings $\bar{E}_a$ (Section 3.1) to F keys and values,

$$\mathbf{K}_a = (\bar{E}_a + \mathbf{T})\mathbf{W}_K, \quad \mathbf{V}_a = (\bar{E}_a + \mathbf{T})\mathbf{W}_V, \quad \mathbf{K}_a, \mathbf{V}_a \in \mathbb{R}^{F \times d_{\text{attn}}}, \quad (3)$$

where $\mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{768 \times d_{\text{attn}}}$ are shared across all accounts and $\mathbf{T} \in \mathbb{R}^{F \times 768}$ is a learned per-feature type embedding that tags each of the F feature series before projection.

The Query Is the Transaction. The same transaction looks different to its two accounts. Its direction, for example, is outgoing for the sender and incoming for the receiver. We write $\mathbf{z}_e$ from each account’s perspective, a sender view $\mathbf{z}_e^u$ and a receiver view $\mathbf{z}_e^v$, and encode both with one shared MLP ϕ into account queries $\mathbf{q}_u = \phi(\mathbf{z}_e^u)$ and $\mathbf{q}_v = \phi(\mathbf{z}_e^v)$ in $\mathbb{R}^{d_{\text{attn}}}$. Each query attends over its own account’s keys (Figure 3a),

$$\text{ctx}_u = \text{Attn}(\mathbf{q}_u, \mathbf{K}_u, \mathbf{V}_u), \quad \text{ctx}_v = \text{Attn}(\mathbf{q}_v, \mathbf{K}_v, \mathbf{V}_v). \quad (4)$$

The keys $\mathbf{K}_a$ and values $\mathbf{V}_a$ are fixed per account while the query changes with every transaction, so two transfers of the same account can weight its feature embeddings differently.

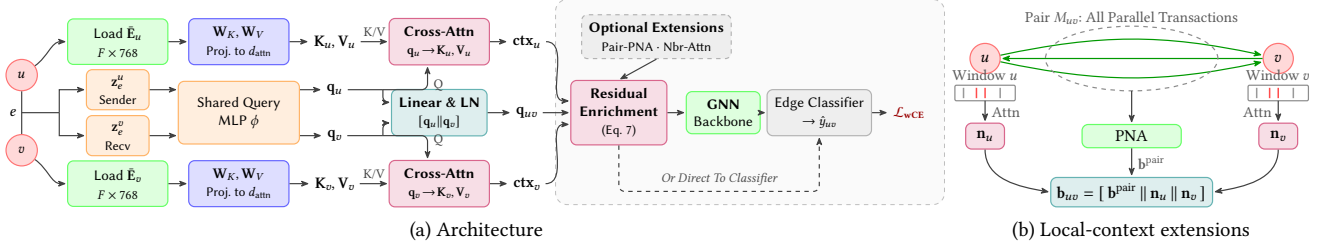


Figure 3: (a) TCEE architecture. The transaction edge $e = (u, v)$ takes each account’s frozen Chronos-2 embedding (green) as keys and values, forms a query from the two transaction views through a shared MLP ϕ , and cross-attends for $\mathbf{ctx}_u, \mathbf{ctx}_v$. Residual enrichment fuses the queries, contexts and optional extensions into $\mathbf{e}_{uv}^{\text{rich}}$, scored directly or after a GNN backbone. The temporal signal enters at the edge, not the node. **(b) The extensions.** Pair pooling aggregates the parallel transactions with PNA, and neighbourhood attention reads each account’s window, giving $\mathbf{b}_{uv}$.

3.3.2 Local Context. The context retrieved in Eq. (4) is rich but coarse, a summary of whole account histories. It is blind to the specific exchange between u and v and to the timing right around the transaction. We add two extensions, one for each blind spot. *Pair pooling* aggregates the transactions between u and v , and *neighbourhood attention* reads the transactions nearest to e in each account’s history (Figure 3b). Both adapt prior GNN components.

Pair Pooling. Let $M_{uv} = \{e' \in E : e' \text{ connects } u \text{ and } v\}$ be the transactions between the pair. We summarise them with PNA [7], which applies four aggregators α to the multiset M , rescales each result by three degree scalars s , and concatenates the twelve outputs,

$$\text{PNA}(M) = \left\|_{\alpha, s} s \alpha(M), \quad \alpha \in \{\text{mean, std, max, min}\}, \quad (5)$$

$$s \in \{1, \lambda, \lambda^{-1}\}, \quad \lambda = \log(1 + |M|) / \mu,$$

where μ , the mean of $\log(1 + \text{deg})$ over the training set, normalises λ to average one. Applied to M_{uv} , with each transaction first encoded from its F raw features by a small MLP, this gives $\mathbf{b}_{uv}^{\text{pair}}$. This is the intra-pair pooling that MEGA-GNN [3] performs as the first stage of its aggregation. The account-level summary includes the pair’s transactions but mixes them into the account’s full history, while the pair pooling extension isolates them.

Neighbourhood Attention. The transactions just before and after the target, in each account’s own history, carry local signal the pooled account embedding averages away. For each account a we attend over $\delta_w(a, e)$, a window of $2w$ transactions around e in a ’s history. Writing t_e for the time of the target edge, the window holds the w most recent transactions before t_e and the first w from t_e onward, the target itself included. If one side holds fewer than w transactions, the window extends on the other side, and a history of at most $2w$ transactions is taken whole. The keys $\mathbf{K}_\delta$ and values $\mathbf{V}_\delta$ are projected from the F raw features of the transactions in $\delta_w(a, e)$. Attention is permutation invariant while the window is an ordered sequence, so we append two features to each transaction as positional encoding relative to e , the time difference from t_e and a flag, 1 for the entries after t_e and 0 otherwise. The account query $\mathbf{q}_a$, projected by $\mathbf{W}_n$, attends over the window,

$$\mathbf{n}_a = \text{Attn}(\mathbf{W}_n \mathbf{q}_a, \mathbf{K}_\delta, \mathbf{V}_\delta), \quad (6)$$

and the outputs for the two accounts concatenate into $\mathbf{b}_{uv}^{\text{nbr}} = [\mathbf{n}_u \parallel \mathbf{n}_v]$. This is one hop on the line graph of G , whose nodes are

the edges of G , adjacent when they share an account. Prior AML work processes this line-graph view with message passing [19]. We use attention instead, restricted to each account’s temporal window. The fixed window keeps it a constant-cost local operation, without the neighbourhood sampling message passing needs. The two extension outputs concatenate into $\mathbf{b}_{uv} = [\mathbf{b}_{uv}^{\text{pair}} \parallel \mathbf{b}_{uv}^{\text{nbr}}]$.

3.3.3 The Enriched Edge. We now fuse the account queries $\mathbf{q}_u, \mathbf{q}_v$, the contexts $\mathbf{ctx}_u, \mathbf{ctx}_v$ retrieved from the edge’s two accounts, and the extension outputs $\mathbf{b}_{uv}$ into one edge representation. The account queries are already encoded by ϕ , so we combine them with one linear map and layer normalisation, $\mathbf{q}_{uv} = \text{LN}(\mathbf{W}_f[\mathbf{q}_u \parallel \mathbf{q}_v])$. The enriched edge representation is then (Figure 3a)

$$\mathbf{e}_{uv}^{\text{rich}} = \text{LN}(\mathbf{W}_r \mathbf{q}_{uv}) + \text{MLP}([\mathbf{q}_{uv} \parallel \mathbf{ctx}_u \parallel \mathbf{ctx}_v \parallel \mathbf{b}_{uv}]), \quad (7)$$

where $\mathbf{W}_r$ is a learned linear map and $\mathbf{b}_{uv}$ the optional extension outputs of Section 3.3.2. The skip keeps the transaction’s own features, encoded in $\mathbf{q}_{uv}$, on a direct path to the output. The contexts enter through the same matching features as at the node (Eq. (2)), $\mathbf{ctx}_u \odot \mathbf{ctx}_v$ and $|\mathbf{ctx}_u - \mathbf{ctx}_v|$.

3.3.4 Classifier and Backbone. TCEE changes only the edge representation, so $\mathbf{e}_{uv}^{\text{rich}}$ drops into any downstream model. TCEE-MLP scores it directly with an MLP, while TCEE-GNN passes it to a GNN backbone. For the GNN backbone we use PNA [7], the aggregator of the best performing baseline variants on most AMLWorld datasets [3, 11]. We name the resulting models TCEE-PNA and Chronos-PNA, which feed the backbone differently. TCEE-PNA injects $\mathbf{e}_{uv}^{\text{rich}}$ at the edge,

$$\mathbf{h}_a^{(0)} = \mathbf{0}, \quad \mathbf{e}_{uv}^{(0)} = \mathbf{W}_E \mathbf{e}_{uv}^{\text{rich}}, \quad (8)$$

with $\mathbf{W}_E$ a learned map. Chronos-PNA injects $\mathbf{h}_a$ at the node,

$$\mathbf{h}_a^{(0)} = \mathbf{W}_C \mathbf{h}_a, \quad \mathbf{e}_{uv}^{(0)} = \mathbf{W}_E \mathbf{z}_e, \quad (9)$$

while the edges carry only projected raw features. Every model is trained end-to-end with the class-weighted cross-entropy

$$\mathcal{L} = - \sum_e w_{y_e} [y_e \log \hat{y}_e + (1 - y_e) \log(1 - \hat{y}_e)], \quad (10)$$

where $\hat{y}_e$ is the illicit probability and the class weight w_{y_e} up-weights the rare illicit class ($y_e=1$), as in the baselines.

3.3.5 Node Classification. On a node task the label belongs to an account, not a transaction. TCEE still enriches every edge (Eq. (8)), the GNN backbone’s message passing pools an account’s incident enriched edges into its node state $\mathbf{h}_a^{(L)}$ after L layers, and a per-node head classifies the account, $\hat{y}_a = \text{MLP}(\mathbf{h}_a^{(L)})$, instead of the edge head. Training uses the loss of Eq. (10) with the sum over accounts.

3.4 Computational Resources

Cost matters in deployment, with banks screening millions of transactions and retraining as criminal behaviour changes. We derive compute and storage costs and compare with the structural models.

3.4.1 Computational Complexity. The frozen encoder runs once per account and split before training. Encoding account a is one transformer forward pass over its $F \times P_a$ grid of patch tokens, so building the store costs $O(F \sum_a (P_a^2 c + P_a c^2))$, with $c=768$ the encoder width. Every model and epoch reuses this store, so the cost is one-time. Training cost depends on what a minibatch holds, a list of edges or a sampled subgraph. Let d be the largest hidden width.

Chronos-MLP and TCEE-MLP train on minibatches of edges. For every edge, both project the two endpoints’ stored embeddings, F vectors of width c per account, down to width d , costing Fcd per account. The projection weights change at every optimiser step, so the result cannot be precomputed and reused. Chronos-MLP reduces the F projected vectors to one account vector (Eq. (1)) and scores the edge with the MLP head, so an epoch over $|E|$ edges costs

$$O(|E| [\underbrace{Fcd}_{\text{pool, Eq. (1)}} + \underbrace{d^2}_{\text{head, Eq. (2)}}]). \quad (11)$$

TCEE-MLP pays the same projection to form its keys and values, then adds three computations. The attention of Eq. (4) scores the transaction’s query against each account’s F keys, pair pooling aggregates the $|M_{uv}|$ transactions between u and v (Eq. (5)), and neighbourhood attention reads a window of $2w$ transactions per account (Eq. (6)). Both extensions first encode each transaction from its F raw features, so one edge costs

$$O(\underbrace{Fcd}_{\text{keys and values}} + \underbrace{Fd}_{\text{account attn.}} + \underbrace{|M_{uv}|Fd}_{\text{pair pool.}} + \underbrace{2wFd}_{\text{nbr. attn.}} + \underbrace{d^2}_{\text{MLPs}}). \quad (12)$$

Only $|M_{uv}|$ varies across edges, so an epoch costs $O(|E| [F(c+1+\overline{|M|}+2w)d+d^2])$, with $\overline{|M|}$ the mean of $|M_{uv}|$ over the training edges. Two things follow. First, no term depends on account degree, account count or a multi-hop neighbourhood, so both models are linear in $|E|$. Second, both epochs are dominated by the shared Fcd projection, since $1+\overline{|M|}+2w \ll c$ and $d \ll Fc$, which is why the two are close in wall-clock (Table 3).

Chronos-PNA and TCEE-PNA train on sampled subgraphs with edges E_s and nodes V_s instead. A subgraph holds far more edges than nodes, so the Fcd projection of an account’s stored embeddings is paid once per node rather than once per edge, $O(|V_s| Fcd)$ per step. The saving does not show because the L -layer PNA backbone ($L=2$) applies Eq. (5) to every sampled neighbour and its $O(L(|E_s|+|V_s|)d^2)$ dominates the step. The baselines MEGA-GNN [3], Multi-GNN [11] and Multi-FraudGT [16] pay this same backbone cost. They score an edge by message passing over a sampled neighbourhood that grows with fanout and depth.

Table 1: Dataset statistics.

Dataset	# Nodes	# Edges	Emb. (GB)	Illicit	Days	Split [%]
HI-Small	515,088	5,078,345	4.7	0.102%	10	64/19/17
LI-Small	705,907	6,924,049	6.5	0.051%	10	64/19/17
HI-Medium	2,077,023	31,898,238	19.1	0.110%	16	61/17/22
LI-Medium	2,032,095	31,251,483	18.7	0.051%	16	61/17/22
HI-Large	2,116,168	179,702,229	19.5	0.124%	97	60/20/20
LI-Large	2,070,980	176,066,557	19.1	0.057%	97	60/20/20
ETH	2,973,489	13,551,303	18.3	0.04%	1261	65/15/20

3.4.2 Storage. Each embedding tensor $\mathbf{E}_a$ holds $768 \cdot F \cdot P_a$ values, so at float16 (2 bytes) the store over N accounts is

$$S = 2 \cdot 768 \cdot F \sum_{a=1}^N P_a \text{ bytes.} \quad (13)$$

An account seen in n_a transactions unfolds into $P_a = \lceil n_a/16 \rceil$ size-16 patches, capped at an 8,192-transaction context (512 patches), so $\sum_a P_a$ grows with the transaction count. Averaging over patches (Section 3.1) sets $P_a=1$, collapsing this to $\bar{\mathbf{E}}_a \in \mathbb{R}^{F \times 768}$ and the store to $2 \cdot 768 \cdot F \cdot N$ bytes, a flat 9 KB per account ($F=6$). Table 1 reports the full graph storage cost, which recurs per split. We keep the store GPU-resident for faster training, and when it exceeds the GPU’s VRAM (medium and large) we hold only the training split resident and memory-map the rest.

4 Experiments

Our experiments answer four research questions, the first three on AML edge classification.

RQ1. How effective (PR-AUC and minority-class F1) is TCEE?

RQ2. What does TCEE cost to train (time-to-best)?

RQ3. How much can the frozen TSFM contribute?

RQ4. How well does TCEE transfer to phishing detection?

4.1 Experimental Setup

4.1.1 Datasets. We use AMLWorld [1], an agent-based transaction simulator with complete labels. It comes in higher- and lower-illicit-ratio regimes (HI, LI) at small, medium and large sizes. We further evaluate on the ETH phishing graph [5, 29], a node- rather than edge-classification task. Table 1 summarises all graphs.

4.1.2 Baselines. The structural floor is GINe and PNAe, the standard GIN [30] and PNA [7] message-passing architectures extended with edge features, as configured in the AML baselines [11]. Strong prior models are MEGA-GNN [3], Multi-FraudGT [16] and Multi-GNN [11]. MEGA-GNN and Multi-GNN each come with a GIN, PNA and GenAgg aggregator. We run all and report per dataset the aggregator with the best test PR-AUC¹, so every reported baseline entry upper-bounds its variants. On the ETH graph, we also compare with the state-of-the-art DIAM [10].

4.1.3 Evaluation. PR-AUC is the primary metric, as it summarises overall performance under heavy class imbalance. We also report minority-class F1, a thresholded metric, for comparability with

¹PNA in most cases, GenAgg for MEGA-GNN on HI-Small and LI-Large and for Multi-GNN on LI-Small.

Table 2: Test PR-AUC and minority-class F1 (%) on the six AMLWorld datasets, mean $\pm$ std over 5 seeds.

Model	HI-Small		LI-Small		HI-Medium		LI-Medium		HI-Large		LI-Large	
	PR-AUC	F1	PR-AUC	F1	PR-AUC	F1	PR-AUC	F1	PR-AUC	F1	PR-AUC	F1
GINe	49.94 $\pm$ 4.02	53.77 $\pm$ 3.39	16.77 $\pm$ 4.27	21.25 $\pm$ 4.86	44.55 $\pm$ 4.53	34.20 $\pm$ 9.98	14.95 $\pm$ 3.42	14.93 $\pm$ 3.45	38.34 $\pm$ 7.04	41.19 $\pm$ 5.65	6.50 $\pm$ 1.77	5.10 $\pm$ 5.30
PNAe	59.40 $\pm$ 3.18	60.93 $\pm$ 3.35	22.29 $\pm$ 1.40	31.26 $\pm$ 2.80	58.44 $\pm$ 3.96	58.12 $\pm$ 4.65	21.04 $\pm$ 2.33	26.99 $\pm$ 3.73	59.90 $\pm$ 0.49	61.37 $\pm$ 0.98	20.81 $\pm$ 1.68	25.48 $\pm$ 3.45
Multi-GNN [11]	69.05 $\pm$ 1.22	49.28 $\pm$ 4.11	30.30 $\pm$ 2.39	34.56 $\pm$ 1.41	74.97 $\pm$ 0.60	73.26 $\pm$ 1.31	35.86 $\pm$ 0.10	43.78 $\pm$ 0.59	73.57 $\pm$ 0.46	72.36 $\pm$ 1.29	27.44 $\pm$ 1.48	33.54 $\pm$ 2.29
MEGA-GNN [3]	74.28 $\pm$ 2.56	73.63 $\pm$ 2.45	38.41 $\pm$ 1.08	45.49 $\pm$ 2.49	78.32 $\pm$ 0.82	78.00 $\pm$ 0.23	39.60 $\pm$ 3.38	48.75 $\pm$ 1.65	77.40 $\pm$ 0.21	76.68 $\pm$ 0.57	29.36 $\pm$ 1.01	34.62 $\pm$ 1.77
Multi-FraudGT [16]	75.46 $\pm$ 0.39	75.68 $\pm$ 0.78	37.72 $\pm$ 0.75	45.20 $\pm$ 1.56	74.88 $\pm$ 0.36	75.97 $\pm$ 0.20	36.15 $\pm$ 0.57	44.66 $\pm$ 0.66	72.62 $\pm$ 0.37	73.48 $\pm$ 0.68	29.16 $\pm$ 0.10	35.52 $\pm$ 0.17
Chronos-MLP	58.92 $\pm$ 3.39	60.04 $\pm$ 1.48	20.16 $\pm$ 3.53	24.05 $\pm$ 7.55	65.15 $\pm$ 1.02	63.50 $\pm$ 4.34	27.30 $\pm$ 1.41	34.13 $\pm$ 1.17	64.89 $\pm$ 1.08	58.83 $\pm$ 1.59	23.79 $\pm$ 0.84	28.51 $\pm$ 0.47
Chronos-PNA	64.89 $\pm$ 2.17	62.22 $\pm$ 2.17	24.92 $\pm$ 3.31	32.39 $\pm$ 2.44	68.20 $\pm$ 1.88	67.73 $\pm$ 2.98	30.89 $\pm$ 1.08	38.25 $\pm$ 0.98	67.85 $\pm$ 1.12*	67.16 $\pm$ 1.44*	26.75 $\pm$ 1.04*	30.45 $\pm$ 2.13*
TCEE-MLP	69.24 $\pm$ 0.61	68.66 $\pm$ 2.02	31.45 $\pm$ 0.93	37.18 $\pm$ 1.05	74.70 $\pm$ 0.53	74.25 $\pm$ 0.66	35.25 $\pm$ 0.36	41.92 $\pm$ 0.41	75.50 $\pm$ 0.96	74.34 $\pm$ 0.90	32.67 $\pm$ 0.77	38.13 $\pm$ 0.95
+ Pair-PNA	75.00 $\pm$ 0.60	73.95 $\pm$ 1.85	34.04 $\pm$ 1.15	40.64 $\pm$ 1.15	79.11 $\pm$ 0.10	78.75 $\pm$ 0.50	37.98 $\pm$ 0.91	44.02 $\pm$ 0.71	80.31 $\pm$ 0.37	79.65 $\pm$ 0.18	38.18 $\pm$ 0.54	43.21 $\pm$ 0.42
+ Nbr-Attn	75.51 $\pm$ 0.87	72.41 $\pm$ 2.46	34.99 $\pm$ 1.28	38.22 $\pm$ 1.93	78.97 $\pm$ 0.31	77.91 $\pm$ 0.58	38.42 $\pm$ 0.87	44.54 $\pm$ 1.30	80.26 $\pm$ 0.07	77.93 $\pm$ 0.31	38.48 $\pm$ 0.18	43.08 $\pm$ 0.37
+ Pair-PNA + Nbr-Attn	78.26 $\pm$ 1.13	76.75 $\pm$ 1.26	34.06 $\pm$ 1.05	40.51 $\pm$ 2.41	81.00 $\pm$ 0.03	80.97 $\pm$ 0.47	41.12 $\pm$ 0.43	46.54 $\pm$ 0.96	81.61 $\pm$ 0.09	81.35 $\pm$ 0.16	40.39 $\pm$ 0.11	45.25 $\pm$ 0.22
TCEE-PNA	74.34 $\pm$ 1.25	73.17 $\pm$ 1.36	31.48 $\pm$ 2.25	36.18 $\pm$ 1.41	78.56 $\pm$ 0.62	77.33 $\pm$ 0.59	40.57 $\pm$ 0.29	45.69 $\pm$ 1.70	78.24 $\pm$ 0.26*	75.03 $\pm$ 0.52*	36.32 $\pm$ 0.72*	39.65 $\pm$ 0.78*
+ Pair-PNA	79.00 $\pm$ 0.66	76.81 $\pm$ 1.40	36.08 $\pm$ 0.83	40.92 $\pm$ 2.24	81.61 $\pm$ 0.13	81.33 $\pm$ 0.25	43.61 $\pm$ 0.41	49.54 $\pm$ 0.33	80.30 $\pm$ 0.50*	77.30 $\pm$ 0.89*	38.69 $\pm$ 0.90*	42.25 $\pm$ 1.13*
+ Pair-PNA + Nbr-Attn	79.94 $\pm$ 0.79	77.88 $\pm$ 1.34	37.32 $\pm$ 1.47	42.69 $\pm$ 2.76	81.81 $\pm$ 0.22	81.55 $\pm$ 0.44	43.74 $\pm$ 0.99	49.91 $\pm$ 0.71	80.50 $\pm$ 0.37*	77.53 $\pm$ 0.45*	38.96 $\pm$ 0.47*	43.16 $\pm$ 0.95*

* Due to computational limitations, those runs were not trained to convergence and were capped at 64 transactions per pair gather for Pair-PNA, so they understate the true performance.

prior AML models [3, 11]. For a controlled comparison our GINe, PNAe and the backbones of Chronos-PNA and TCEE-PNA use the hyperparameters of Multi-GNN and MEGA-GNN [3, 11]. We tune our models’ learning rate, attention window size and the widths d_{proj} and d_{attn} on validation PR-AUC. Following the baselines [3, 11, 16], which message-pass over all edges in a split, embeddings are computed over each full split. Each run trains on an NVIDIA A40 GPU with early stopping within a fixed epoch budget, so quality and cost stay comparable. We report each test metric at the checkpoint that maximises its validation score, averaged over 5 seeds. Cost is *time-to-best*, the wall-clock time to the best validation PR-AUC.

4.2 Experimental Results

4.2.1 Effectiveness (RQ1). We discuss Table 2 in three steps.

First, the frozen Chronos-2 embeddings are strong on their own. Chronos-MLP, with no message passing, beats the GINe structural floor on all AML datasets and PNAe on the medium and large graphs, reaching 65.15 PR-AUC on HI-Medium against PNAe’s 58.44. A PNA backbone lifts Chronos-PNA higher, yet both Chronos models trail Multi-GNN and MEGA-GNN, as node-level fusion hands every transaction of an account the same vector.

Second, building the representation per transaction adds more. TCEE-MLP consumes the same embeddings as Chronos-MLP, raising PR-AUC on every dataset, 28.5% on average.

Third, the two extensions help. With both on, the full TCEE-MLP beats the strongest baseline of each dataset in PR-AUC on five of the six, with margins averaging 10.8% and reaching 37.6% on LI-Large. It also leads in minority-class F1 on four of those five, trailing MEGA-GNN on LI-Medium. Adding the PNA backbone raises PR-AUC a further 4.8% on the datasets trained to convergence, and 2.4% on average across all six. TCEE-MLP trails only on LI-Small, which holds the fewest illicit transactions of any AML dataset. There the full TCEE-PNA narrows the gap to within one standard deviation, 37.3 PR-AUC against MEGA-GNN’s 38.4.

4.2.2 Efficiency (RQ2). Table 3 reports time-to-best including the one-time extraction all models reuse. The plain TCEE-MLP reaches its best score 7.7 $\times$ faster end-to-end on average than the strongest

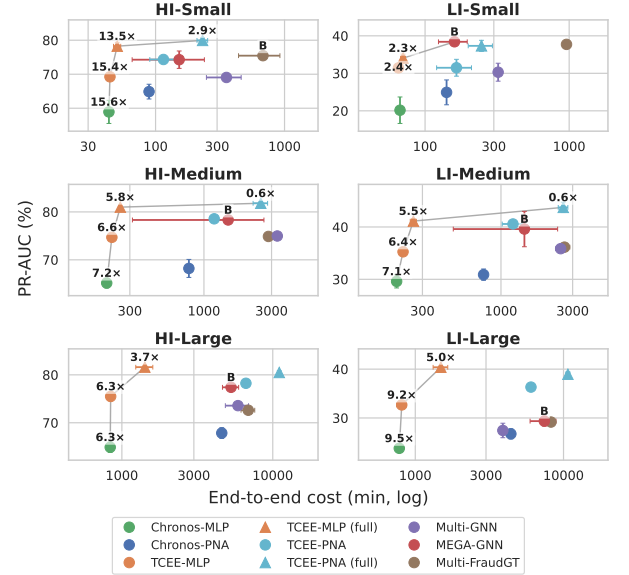


Figure 4: Test PR-AUC (%) against end-to-end cost (log minutes). Top left is better, the grey line the Pareto front, and $\times N$ each model’s speed-up over the strongest baseline B.

baseline, and the full variant 6.0 $\times$ faster. TCEE-MLP adds 8.0 to 11.3 PR-AUC points over Chronos-MLP at almost no extra cost.

Figure 4 plots quality against cost, with better models at the top left. The Chronos models are cheap but score lowest, the structural baselines score well but cost the most, and the TCEE-MLP variants are cheap and score well, holding every dataset’s Pareto front. The GNN backbone improves PR-AUC but trains 7.0 $\times$ longer end-to-end than the full TCEE-MLP where trained to convergence, and 7.1 $\times$ across all six datasets.

4.2.3 Ablation Studies (RQ3). We ablate the frozen TSFM’s input and pooling (Table 4). On HI-Small and LI-Small, removing the

Table 3: Extraction and end-to-end wall-clock cost in minutes. End-to-end includes extraction.

Model	HI-S	LI-S	HI-M	LI-M	HI-L	LI-L
Extraction	39	58	177	181	523	509
Chronos-MLP	43 $\pm$ 2	67 $\pm$ 4	206 $\pm$ 8	202 $\pm$ 3	837 $\pm$ 69	780 $\pm$ 67
Chronos-PNA	89 $\pm$ 11	141 $\pm$ 20	783 $\pm$ 47	766 $\pm$ 80	4592 $\pm$ 472	4414 $\pm$ 220
TCEE-MLP	44 $\pm$ 1	65 $\pm$ 2	224 $\pm$ 7	223 $\pm$ 11	843 $\pm$ 49	810 $\pm$ 62
+ Pair-PNA	47 $\pm$ 2	68 $\pm$ 3	259 $\pm$ 14	269 $\pm$ 16	1344 $\pm$ 135	1453 $\pm$ 118
+ Pair-PNA + Nbr-Attn	50 $\pm$ 3	70 $\pm$ 3	257 $\pm$ 4	261 $\pm$ 12	1420 $\pm$ 184	1488 $\pm$ 168
TCEE-PNA	115 $\pm$ 25	165 $\pm$ 45	1181 $\pm$ 83	1202 $\pm$ 183	6639 $\pm$ 260	6032 $\pm$ 260
+ Pair-PNA + Nbr-Attn	232 $\pm$ 22	246 $\pm$ 47	2517 $\pm$ 295	2597 $\pm$ 191	11041 $\pm$ 113	10706 $\pm$ 325
Baselines						
Multi-GNN	354 $\pm$ 107	321 $\pm$ 15	3294 $\pm$ 143	2497 $\pm$ 187	5880 $\pm$ 1030	3897 $\pm$ 45
MEGA-GNN	152 $\pm$ 87	160 $\pm$ 36	1482 $\pm$ 1169	1432 $\pm$ 951	5291 $\pm$ 650	7422 $\pm$ 1474
Multi-FraudGT	680 $\pm$ 241	958 $\pm$ 52	2852 $\pm$ 38	2648 $\pm$ 226	6879 $\pm$ 716	8231 $\pm$ 261

Table 4: Ablations (PR-AUC %). F feature series, P patches.

Ablation	Setting	HI-Small	LI-Small
Counterparty	Chronos-MLP, -CP	54.69 $\pm$ 4.15	18.80 $\pm$ 3.16
	Chronos-MLP, +CP	58.92 $\pm$ 3.39	20.16 $\pm$ 3.53
	TCEE-MLP, -CP	63.20 $\pm$ 2.84	27.64 $\pm$ 0.83
	TCEE-MLP, +CP (default)	69.24 $\pm$ 0.61	31.45 $\pm$ 0.93
h_a pool	Chronos-MLP, mean over F	58.74 $\pm$ 4.32	19.98 $\pm$ 2.29
	Chronos-MLP, attn. over F (default)	58.92 $\pm$ 3.39	20.16 $\pm$ 3.53
Pooling (TCEE-MLP)	Mean over F and P	67.24 $\pm$ 1.58	30.84 $\pm$ 0.62
	Attn. over F , PNA over P	68.13 $\pm$ 1.51	30.54 $\pm$ 1.37
	Attn. over F and P	69.13 $\pm$ 0.78	29.74 $\pm$ 1.38
	Attn. over F , mean over P (default)	69.24 $\pm$ 0.61	31.45 $\pm$ 0.93
Context (TCEE-MLP)	-Chronos	55.01 $\pm$ 0.87	23.63 $\pm$ 0.81
	+Chronos (default)	69.24 $\pm$ 0.61	31.45 $\pm$ 0.93

Table 5: ETH phishing node classification, PR-AUC and minority-class F1 (%).

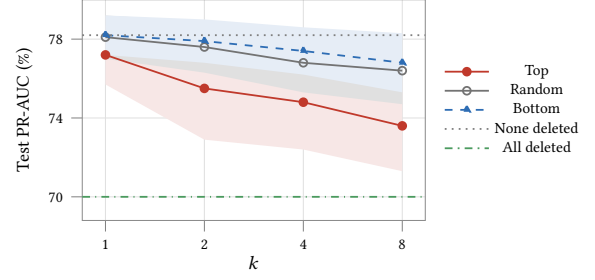
Model	PR-AUC	F1
Multi-GNN [11]	61.2 $\pm$ 0.6	65.0 $\pm$ 1.1
MEGA-GNN [3]	60.5 $\pm$ 4.5	62.7 $\pm$ 5.8
DIAM [10]	55.3 $\pm$ 1.8	64.6 $\pm$ 1.4
Chronos-MLP	31.4 $\pm$ 0.7	26.0 $\pm$ 2.0
Chronos-PNA	70.1 $\pm$ 0.9	75.5 $\pm$ 1.3
TCEE-PNA	70.9 $\pm$ 1.1	75.4 $\pm$ 0.8
+ Pair-PNA + Nbr-Attn	72.1 $\pm$ 0.9	73.8 $\pm$ 1.2

counterparty index costs Chronos-MLP 4.2 and 1.4 PR-AUC points and TCEE-MLP 6.0 and 3.8, so structure is informative.

Attending over an account’s F feature embeddings helps only when the query is the transaction. Chronos-MLP’s attention over F matches a plain mean, while in TCEE-MLP it adds 2.0 and 0.6 points.

Along the patch axis, PNA and attention over both F and P stay within two points of the plain mean and win nowhere, as about 63% of accounts fit in a single 16-transaction patch. Longer medium and large histories might reward richer pooling, but the per-patch store outgrows the GPU (Section 3.4). Finally, we remove the Chronos context from TCEE-MLP entirely. With no keys or values in Eq. (3) there is nothing to attend over, and the model reduces to an MLP over the fused transaction queries q_{uv} . This costs 14.2 and 7.8 points, the largest drop of any ablation.

4.2.4 Phishing Detection (RQ4). For the node task, we evaluate phishing detection on the ETH graph against Multi-GNN [11], MEGA-GNN [3] and DIAM [10]. TCEE-MLP scores transactions

**Figure 5: Test PR-AUC on HI-Small after deleting the k top, bottom or random attended window transactions per end-point.**

rather than accounts, so we instead use TCEE-PNA with a per-node head (Section 3.3.5). Chronos-MLP classifies each account from its pooled h_a alone and reaches only 31.4 PR-AUC, possibly because ETH carries four features against AMLWorld’s six. With a PNA backbone, the same embeddings excel, Chronos-PNA and TCEE-PNA scoring 14.5% and 15.9% above the strongest baseline in PR-AUC (Table 5). The two match because message passing pools an account’s edges into one node state, averaging out the distinctions TCEE builds. On this task, edge-level querying on its own shows no measurable benefit over node-level fusion. The two extensions do add little on PR-AUC, with the full TCEE-PNA reaching 72.1 PR-AUC, 17.8% above the strongest baseline, but give slightly worse F1 than plain TCEE-PNA.

5 Discussion

TCEE models both structure and time, and without a backbone it beats every evaluated baseline on five of six AMLWorld datasets in PR-AUC at lower end-to-end cost. We discuss the implications and the next steps.

Deployment. Transaction monitoring raises alerts for investigators, and 90 to 95% of those from rule-based systems are false positives [18]. Investigators work through the alerts in order, so a better ranking saves review time. Even on the low-illicit datasets TCEE reaches up to 43.7 PR-AUC, against 0.059 for a random ranking. It can support an analyst, not replace one. Cost matters because laundering methods change, requiring frequent retraining [18]. On the large graphs the full TCEE-MLP finishes in about a day on one A40 GPU, where the structural baselines need two to six days (Table 3). How often banks retrain is not public, so whether this suffices remains open.

Generalisation. AMLWorld is generated to mirror real transaction behaviour [1]. Whether these results hold on real bank data, where labels are incomplete, should be explored. AMLWorld also covers fixed typologies, most within two hops. TCEE-MLP sees one hop and the PNA backbone two, improving PR-AUC where trained to convergence.

Interpretability. Each prediction exposes its attention over both accounts’ F feature series and the 2w surrounding transactions. An analyst can point to this trace, whereas message-passing detectors need separate explainers [15]. However, attention marks what the

model consulted, not why it decided [13, 27]. We test whether the neighbourhood attention weights are informative on HI-Small by deleting window transactions in order of their attention weight and measuring the drop in PR-AUC (Figure 5). Removing the top 8 transactions per account causes over half of the drop from removing the whole window, against about a fifth for 8 random transactions.

Limitations and Future Work. These findings are limited by one GPU, one frozen encoder, and one domain. Larger GPUs can hold the medium and large embeddings resident, making TCEE faster. Future work can try other TSFM encoders, or fine-tune one. TCEE is also not tied to financial crime. It fits any multigraph whose nodes carry a history of timed events, such as cybersecurity data.

6 Conclusion

Financial crime detection requires understanding not only the structure of financial networks, but also how the behaviour of accounts evolves over time. Our work shows that combining structural and temporal information can improve detection considerably, and that the gains depend on how this information is used in the model. When an account is represented the same way in all of its transactions, those transactions can become hard to tell apart. TCEE instead builds a separate representation for every transaction from its own features, its accounts' histories and its local context. This design surpasses the strongest evaluated baseline on five of six anti-money laundering datasets while training 6× faster. Adding a GNN that captures broader structure yields further gains on the smaller datasets, and the same combination also outperforms all evaluated baselines on Ethereum phishing detection. Together, these results suggest a simple lesson for detecting illicit transactions: bring the context directly to the point where the decision is made, the individual transaction.

References

- [1] Erik Altman, Jovan Blanuša, Luc Von Niederhäusern, Béni Egressy, Andreea Anghel, and Kubilay Atasü. 2023. Realistic synthetic financial transactions for anti-money laundering models. *Advances in Neural Information Processing Systems* 36 (2023), 29851–29874.
- [2] Abdul Fatir Ansari, Oleksandr Shchur, Jaris Küken, Andreas Auer, Boran Han, Pedro Mercado, Syama Sundar Rangapuram, Huibin Shen, Lorenzo Stella, Xiyuan Zhang, Mononito Goswami, Shubham Kapoor, Danielle C. Maddix, Pablo Gueron, Tony Hu, Junming Yin, Nick Erickson, Prateek Mutalik Desai, Hao Wang, Huzefa Rangwala, George Karypis, Yuyang Wang, and Michael Bohlke-Schneider. 2025. Chronos-2: From univariate to universal forecasting. *arXiv preprint arXiv:2510.15821* (2025).
- [3] H Çağrı Bilgi, Lydia Y Chen, and Kubilay Atasü. 2024. Multigraph Message Passing with Bi-Directional Multi-Edge Aggregations. *arXiv preprint arXiv:2412.00241* (2024).
- [4] Jovan Blanuša, Maximo Cravero Baraja, Andreea Anghel, Luc Von Niederhäusern, Erik Altman, Haris Pozidis, and Kubilay Atasü. 2024. Graph Feature Preprocessor: Real-time Subgraph-based Feature Extraction for Financial Crime Detection. In *Proceedings of the 5th ACM International Conference on AI in Finance*. ACM, 9 pages. doi:10.1145/3677052.3698674
- [5] Liang Chen, Jiaying Peng, Yang Liu, Jintang Li, Fenfang Xie, and Zibin Zheng. 2019. XBLOCK Blockchain Datasets: InPlusLab Ethereum Phishing Detection Datasets. <http://xblock.pro/ethereum/>.
- [6] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 conference on empirical methods in natural language processing*. 670–680.
- [7] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. 2020. Principal neighbourhood aggregation for graph nets. *Advances in neural information processing systems* 33 (2020), 13260–13271.
- [8] Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. 2024. A decoder-only foundation model for time-series forecasting. In *International Conference on Machine Learning*.
- [9] Cameron Diao and Ricky Loynd. 2023. Relational attention: Generalizing transformers for graph-structured tasks. In *International Conference on Learning Representations*.
- [10] Zhihao Ding, Jieming Shi, Qing Li, and Jiannong Cao. 2024. Effective illicit account detection on large cryptocurrency multigraphs. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 457–466.
- [11] Béni Egressy, Luc Von Niederhäusern, Jovan Blanuša, Erik Altman, Roger Wattenhofer, and Kubilay Atasü. 2024. Provably powerful graph neural networks for directed multigraphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 11838–11846.
- [12] Steve Gounoue, Ashutosh Sao, and Simon Gottschalk. 2025. TeMP-TraG: Edge-based Temporal Message Passing in Transaction Graphs. *arXiv preprint arXiv:2503.16901* (2025).
- [13] Sarthak Jain and Byron C Wallace. 2019. Attention is not explanation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*. 3543–3556.
- [14] Utku Gökrem Ketenci, Tolga Kurt, Selim Önal, Cenk Erbil, Sinan Aktürkkoğlu, and Hande Şerban İlhan. 2021. A time-frequency based suspicious activity detection for anti-money laundering. *IEEE Access* 9 (2021), 59957–59967.
- [15] Kaidi Li, Tianmeng Yang, Min Zhou, Jiahao Meng, Shendi Wang, Yihui Wu, Boshuai Tan, Hu Song, Lujia Pan, Fan Yu, Zhenli Sheng, and Yunhai Tong. 2024. SEFraud: Graph-based self-explainable fraud detection via interpretative mask learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5329–5338.
- [16] Junhong Lin, Xiaojie Guo, Yada Zhu, Samuel Mitchell, Erik Altman, and Julian Shun. 2024. FraudGT: A simple, effective, and efficient graph transformer for financial fraud detection. In *Proceedings of the 5th ACM International Conference on AI in Finance (ICAIF '24)*. ACM, 292–300. doi:10.1145/3677052.3698648
- [17] Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2023. A time series is worth 64 words: Long-term forecasting with transformers. In *International Conference on Learning Representations*.
- [18] Berkan Oztas, Deniz Cetinkaya, Festus Adedoyin, Marcin Budka, Gokhan Aksu, and Huseyin Dogan. 2024. Transaction monitoring in anti-money laundering: A qualitative analysis and points of view from industry. *Future Generation Computer Systems* 159 (2024), 161–171. doi:10.1016/j.future.2024.05.027
- [19] Chung-Hoo Poon, James Kwok, Calvin Chow, and Jang-Hyeon Choi. 2025. LineMVGNN: Anti-money laundering with line-graph-assisted multi-view graph neural networks. *AI* 6, 4 (2025), 69.
- [20] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637* (2020).
- [21] David Savage, Qingmai Wang, Pauline Chou, Xiuzhen Zhang, and Xinghuo Yu. 2016. Detection of money laundering groups using supervised learning in networks. *arXiv preprint arXiv:1608.00708* (2016).
- [22] Zezhi Shao, Zhao Zhang, Fei Wang, and Yongjun Xu. 2022. Pre-training enhanced spatial-temporal graph neural network for multivariate time series forecasting. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining (KDD '22)*. ACM, 1567–1577. doi:10.1145/3534678.3539396
- [23] Michele Starnini, Charalampos E Tsourakakis, Maryam Zamanipour, André Panisson, Walter Allasia, Marco Fornasier, Laura Li Puma, Valeria Ricci, Silvia Ronchiadri, Angela Ugrinoska, Marco Varetto, and Dario Moncalvo. 2021. Smurf-based anti-money laundering in time-evolving transaction networks. In *Machine Learning and Knowledge Discovery in Databases*. Springer-Verlag, 171–186. doi:10.1007/978-3-030-86514-6_11
- [24] United Nations Office on Drugs and Crime. 2024. Money Laundering Overview. <https://www.unodc.org/unodc/en/money-laundering/overview.html>.
- [25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
- [26] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I. Weidele, Claudio Bellei, Tom Robinson, and Charles E. Leiserson. 2019. Anti-Money Laundering in Bitcoin: Experimenting with Graph Convolutional Networks for Financial Forensics. *arXiv preprint arXiv:1908.02591* (2019).
- [27] Sarah Wiegrefe and Yuval Pinter. 2019. Attention is not not explanation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. 11–20.
- [28] Gerald Woo, Chenghao Liu, Akshat Kumar, Caiming Xiong, Silvio Savarese, and Doyen Sahoo. 2024. Unified training of universal time series forecasting transformers. In *International Conference on Machine Learning*.
- [29] Jiajing Wu, Qi Yuan, Dan Lin, Wei Yu, Wei Chen, Chuan Chen, and Zibin Zheng. 2019. Who are the phishers? Phishing scam detection on Ethereum via network embedding. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52, 2 (2019), 1156–1166.
- [30] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2019).

4

Further Discussion

This thesis investigated whether Chronos-2 representations of account transaction histories can improve financial crime detection. The results are promising but require careful interpretation. Full TCEE-MLP outperformed the strongest evaluated baseline on five of six AMLWorld datasets and trained much faster. PNA usually improved it further, while history representations combined with graph learning also performed well on Ethereum phishing detection.

4.1. Interpreting the Main Results

Chronos accounts for much of TCEE’s improvement. The results for RQ1 show that account histories add useful information to structural baselines. While not state-of-the-art performance, Chronos-MLP and Chronos-PNA give competitive results given their low training cost. The model we proposed, plain TCEE outperformed Chronos-MLP and Chronos-PNA on every AMLWorld dataset, and the extensions usually helped further. The transaction-specific part of TCEE explains only part of its improvement. When Chronos-MLP was given the same sender and receiver transaction views used to build TCEE’s queries, it recovered most of the performance gap to TCEE (Appendix B.2.3). Replacing TCEE’s feature attention with a simple mean also caused only a small drop in performance (Appendix B.5.3), and part of this difference was within the variation across seeds. Together, these results suggest that the Chronos-2 embeddings are responsible for most of TCEE’s improvement. Representing each transaction differently for the sender and receiver adds some further improvement, while the feature attention mechanism adds only a small amount.

Chronos-2 matters less when other information is available. The results for RQ3 show that the Chronos-2 embeddings make a substantial contribution, but how large that contribution is depends on the dataset and what other inputs the model has. Removal of Chronos context from plain TCEE caused a large loss on every dataset (Appendix B.1.1). Once the pair and neighbourhood extensions are both active, they supply local context of their own and the frozen representation adds little. The account histories (Chronos-2 embeddings) therefore matter most where the model has no other source of context. Notably, these representations are not purely temporal. The histories also contain transaction attributes such as direction and counterparty index. On both Small datasets, removal of the counterparty index caused a larger loss than removal of the time delta (Appendix B.2.1). Removal of both was worst. Time helps, but it is not the only important source of information.

Pretraining and model capacity are not always necessary. Pretraining and model capacity did not help in every setting. On the Small datasets, pretrained Chronos-2 clearly outperformed the same architecture with randomly initialised weights (Appendix B.1.2). On the Medium and Large datasets, the two versions performed similarly, and Chronos-2 without pretrained weights sometimes scored slightly higher. This suggests that the encoder architecture itself can produce useful representations without pretraining. Pretraining may be more useful when labelled data are scarce, although the current

evidence does not establish that data volume causes this difference. Wider projections and separate parameters had no consistent advantage past a certain point (Appendix B.5).

Adding a PNA backbone usually improves TCEE. TCEE can also be combined with any GNN backbone that takes edge features. Combining plain TCEE with a PNA backbone, PR-AUC improved on all datasets apart from on LI-Small. The reason for this exception should be investigated further. Large dataset PNA results remain lower bounds because the runs did not converge (Appendix A.5). These results suggest that adding a PNA backbone is generally beneficial to TCEE.

Component effectiveness varies across datasets. The contribution of TCEE's components also varied across datasets. Combining pair and neighbourhood context generally improved performance, but not on LI-Small, where neighbourhood attention alone performed slightly better (Appendix B.1.1). This was also the only dataset where PNA did not clearly help on plain TCEE. LI-Large, by contrast, showed particularly large gains over MEGA-GNN. The current experiments cannot determine whether these differences arise from dataset characteristics, laundering patterns, or optimisation. Further analysis by typology and graph structure is needed to understand when and why each component is most useful.

4.2. Efficiency and Applicability

The results for RQ2 show that TCEE offers a good balance between detection performance and training cost. Full TCEE-MLP was about six times faster on average than the strongest baseline, even when TSFM embedding extraction was included. On the Large datasets, training took roughly one day on a single NVIDIA A40, while the strongest structural alternatives required several days. Faster training matters because transaction monitoring models need regular updates as behaviour and laundering strategies change. Whether this is fast enough in practice will depend on the size of the transaction data, the available hardware, and how often the model needs to be retrained. The reported cost also covers training to the best checkpoint, but not prediction time or embedding updates.

The effectiveness results also need to be considered from an operational perspective. Full TCEE-MLP also scored a higher minority-class F1 than the strongest evaluated baseline on four of the six AMLWorld datasets. However, banks may value precision and recall differently depending on their risk tolerance and investigation capacity. The practical value of TCEE will therefore depend on whether its precision and recall values meet each bank's operational requirements when tested on real transaction data and realistic alert volumes.

4.3. Interpretability and Actionability

4.3.1. Feature Attention

Feature attention provides limited insight into why the model reaches a decision. When testing the weight variation between transactions, more than 92% of the measured variation was associated with transactions belonging to different accounts rather than different transactions belonging to the same account (Appendix C.2.2). Transactions involving the same account therefore received very similar attention weights. This suggests that feature attention was mainly account specific rather than strongly adjusted for each transaction.

The attention weights did not provide a reliable ranking of feature importance. When features were ranked by their average attention weight, currency usually appeared near the top and direction near the bottom. The positions of the other features changed across seeds, and there was no consistent difference between licit and illicit transactions (Appendix C.2.3). However, removing each feature from the model produced a different ordering. Payment type and counterparty index caused the largest drops in performance, while removing direction had no measurable effect (Appendix C.3.1). The same limitation appeared for individual predictions. Removing the most attended features harmed licit predictions more than random removal, but this was not the case for illicit predictions (Appendix C.3.2). Feature attention should therefore not be treated as a faithful explanation of the model's decision.

4.3.2. Neighbourhood Attention

Neighbourhood attention has greater value as a way to find relevant context. Removal of highly attended surrounding transactions hurt more than removal of random or low attention transactions (Appendix C.3.3). The model also placed somewhat more weight on transactions from the same laundering pattern than temporal proximity alone could explain. This effect was modest, varied across seeds, and weakened when few related transactions appeared in the window (Appendix C.4.1). Neighbourhood attention may help an analyst find related activity in some cases, but its weights should not be treated as a complete explanation of the model's decision.

4.4. Generalisability and Transfer Beyond AML

The findings for RQ4 give early evidence of transfer beyond AML. On a real Ethereum graph, Chronos-PNA and TCEE-PNA outperformed the evaluated models. Chronos-MLP performed poorly. The benefit therefore appears to depend on TSFM representations combined with graph learning for this task. One dataset, one node classification task, and one TSFM however cannot establish broad transfer.

Generalisability to real banking data remains a central open question. AMLWorld is synthetic, although designed to reproduce realistic transaction behaviour. Real bank labels are incomplete, delayed, and may be biased towards cases detected by existing monitoring systems. AMLWorld also contains eight fixed typologies, so the results do not show how TCEE would respond to unseen laundering strategies. Beyond finance, cybersecurity is a plausible application because interactions such as logins, messages, and network connections form temporal graphs in which each user or device has a history of events. Separate studies are needed to determine TCEE's effectiveness in other domains.

4.5. Societal Impact and Ethics

Financial crime causes real harm, from laundered proceeds that fund trafficking and corruption to phishing attacks that defraud individual victims. At the same time, rule based monitoring systems often produce large numbers of false alerts, which consume investigator time and place legitimate customers under review. Since TCEE improves alert ranking, it could help investigators reach illicit activity earlier. However, the experiments do not show whether it would reduce false alerts at the operating thresholds used by banks.

Showing the surrounding transactions linked to an alert could make investigations more focused and transparent. Neighbourhood attention may help investigators identify related activity and understand which context the model emphasised. This is important because false alerts can expose legitimate customers to unnecessary review or restrictions. However, attention weights do not fully explain the model's decision and should never be used alone to justify action against a customer.

The account history representations used by TCEE also require careful protection. They are derived from personal transaction data and describe patterns in an account's behaviour, so they should be treated as sensitive information. Detection scores should only be used to prioritise alerts for human investigators and should not trigger automatic action against customers. Institutions should also monitor false alerts and possible differences in performance across customer groups. The shorter training time of TCEE may reduce the computation needed for regular retraining, although energy consumption was not measured directly.

4.6. Limitations

The main limitation concerns generalisability beyond the evaluated datasets. Most AML evidence comes from one synthetic data generator. AMLWorld provides complete labels and known patterns, but it cannot represent every property of a real banking system. It also contains eight fixed typologies, so the results do not show how TCEE would respond to unseen laundering strategies. The Ethereum experiment provides evidence from real data, but it concerns phishing and uses node classification rather than transaction classification. Additionally, only Chronos-2 was evaluated as a TSFM. The findings may therefore differ with another encoder.

A second limitation is that the experimental coverage was not equal across all datasets and models. Several explanatory experiments focused on the Small datasets, while all analyses of the attention

weights were conducted on HI-Small (Appendix C). Some TCEE-PNA models on the Large datasets did not reach convergence because of computational constraints (Appendix A.5) and Pair-PNA histories for those were capped (Appendix A.3). The reported PNA results on the Large datasets should therefore be treated as lower bounds. The attention findings should also not be assumed to hold across every dataset or laundering pattern.

The efficiency evaluation covers only part of the cost of using TCEE. The reported runtime includes embedding extraction and training to the best checkpoint, but it does not include prediction time or the cost of updating embeddings as account histories change. The results therefore show that TCEE trains faster than the evaluated baselines, but they do not establish its total cost in production.

4.7. Future Research

Further research should explore why TCEE performs better on some datasets than on others. Performance could be separated by laundering typology, account history length, pair size, and illicit ratio. Comparing LI-Small and LI-Large may help understand when each TCEE component is most useful. The PNA models on the Large datasets should also be trained to convergence in order to assess their full potential contribution. Other TSFM encoders and selective fine-tuning should be tested to determine whether the results depend on the use of a frozen Chronos-2.

Broader evaluation is also needed to understand how well TCEE generalises. This should include real banking data, previously unseen laundering strategies, and incomplete labels. TCEE could also be tested in other domains such as cybersecurity. These experiments would show whether TCEE captures useful temporal patterns beyond the datasets and financial crime tasks considered in this thesis.

4.8. Concluding Remarks

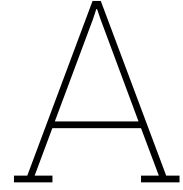
This thesis addressed four research questions. For RQ1, full TCEE-MLP improved PR-AUC over the strongest evaluated baseline on five of six AMLWorld datasets, with a 10.8% mean improvement on those five. Adding a PNA backbone gave a further 4.8% improvement over full TCEE-MLP on the four datasets trained to convergence. For RQ2, full TCEE-MLP reached its best validation performance 6.0× faster on average than the strongest baseline, including the one-time cost of Chronos-2 embedding extraction. For RQ3, adding frozen Chronos-2 context to plain TCEE-MLP improved PR-AUC by 33.4% on average across the six AMLWorld datasets, although its marginal contribution decreased once the extensions were added. For RQ4, TSFM representations combined with graph learning performed well on Ethereum phishing detection. Chronos-PNA and full TCEE-PNA exceeded the strongest baseline by 14.5% and 17.8% in PR-AUC, respectively.

The central conclusion of this thesis is that Chronos-2 representations of account transaction histories are useful for financial crime detection. TCEE provides a way to use these account representations directly to classify individual transactions. These findings open opportunities to study other TSFM encoders, fine-tuning, and applications in domains such as cybersecurity. Overall, the results show that using TSFMs to represent account behaviour is a direction worth pursuing in financial crime detection.

References

- [1] United Nations Office on Drugs and Crime. *Money Laundering Overview*. <https://www.unodc.org/unodc/en/money-laundering/overview.html>. 2024.
- [2] Federal Bureau of Investigation. *2025 IC3 Annual Report*. https://www.ic3.gov/AnnualReport/Reports/2025_IC3Report.pdf. Internet Crime Complaint Center; \$20.877B reported losses, \$11.366B with a cryptocurrency nexus. 2026.
- [3] Berkan Oztas et al. "Transaction monitoring in anti-money laundering: A qualitative analysis and points of view from industry". In: *Future Generation Computer Systems* 159 (2024), pp. 161–171. DOI: 10.1016/j.future.2024.05.027.
- [4] Erik Altman et al. "Realistic synthetic financial transactions for anti-money laundering models". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 29851–29874.
- [5] Zhihao Ding et al. "Effective illicit account detection on large cryptocurrency multigraphs". In: *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 2024, pp. 457–466.
- [6] Utku Gökrem Ketenci et al. "A time-frequency based suspicious activity detection for anti-money laundering". In: *IEEE Access* 9 (2021), pp. 59957–59967.
- [7] Béni Egressy et al. "Provably powerful graph neural networks for directed multigraphs". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 2024, pp. 11838–11846.
- [8] H Çağrı Bilgi, Lydia Y Chen, and Kubilay Atasü. "Multigraph Message Passing with Bi-Directional Multi-Edge Aggregations". In: *arXiv preprint arXiv:2412.00241* (2024).
- [9] Junhong Lin et al. "FraudGT: A simple, effective, and efficient graph transformer for financial fraud detection". In: *Proceedings of the 5th ACM International Conference on AI in Finance*. ICAIF '24. ACM, 2024, pp. 292–300. DOI: 10.1145/3677052.3698648.
- [10] Emanuele Rossi et al. "Temporal graph networks for deep learning on dynamic graphs". In: *arXiv preprint arXiv:2006.10637* (2020).
- [11] Abdul Fatir Ansari et al. "Chronos-2: From univariate to universal forecasting". In: *arXiv preprint arXiv:2510.15821* (2025).
- [12] Weijian Li et al. "StarEmbed: benchmarking time series foundation models on astronomical observations of variable stars". In: *arXiv preprint arXiv:2510.06200* (2025).
- [13] Justin Gilmer et al. "Neural Message Passing for Quantum Chemistry". In: *Proceedings of the 34th International Conference on Machine Learning*. arXiv:1704.01212. 2017.
- [14] Keyulu Xu et al. "How powerful are graph neural networks?" In: *arXiv preprint arXiv:1810.00826* (2019).
- [15] Gabriele Corso et al. "Principal neighbourhood aggregation for graph nets". In: *Advances in neural information processing systems* 33 (2020), pp. 13260–13271.
- [16] Qimai Li, Zhichao Han, and Xiao-Ming Wu. "Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning". In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. 2018.
- [17] Guillaume Jaume et al. "edGNN: A Simple and Powerful GNN for Directed Labeled Graphs". In: *International Conference on Learning Representations*. arXiv:1904.08745. 2019.
- [18] Jiaxuan You et al. "Identity-Aware Graph Neural Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 2021, pp. 10737–10745.
- [19] Weihua Hu et al. "Strategies for Pre-training Graph Neural Networks". In: *International Conference on Learning Representations*. arXiv:1905.12265. 2020.

- [20] Abdul Fatir Ansari et al. "Chronos: Learning the Language of Time Series". In: (). arXiv: 2403.07815 [cs.LG].
- [21] Yuxuan Liang et al. "Foundation Models for Time Series Analysis: A Tutorial and Survey". In: *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. arXiv:2403.14735. 2024.
- [22] Gerald Woo et al. "Unified training of universal time series forecasting transformers". In: *International Conference on Machine Learning*. 2024.
- [23] Abhimanyu Das et al. "A decoder-only foundation model for time-series forecasting". In: *International Conference on Machine Learning*. 2024.
- [24] Mononito Goswami et al. "MOMENT: A Family of Open Time-series Foundation Models". In: *International Conference on Machine Learning*. arXiv:2402.03885. 2024.
- [25] Yuqi Nie et al. "A time series is worth 64 words: Long-term forecasting with transformers". In: *International Conference on Learning Representations*. 2023.
- [26] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).
- [27] Juho Lee et al. "Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning*. 2019.



Dataset Statistics and Experimental Details

In this appendix we document statistics of the six AMLWorld datasets, including their laundering typologies, and local structures, and list details of the experimental setup.

For an account a , the history length L_a counts all transactions incident to a before the evaluation time of the relevant split. The pair multiset M_{uv} contains all transactions between u and v in either direction.

A.1. Dataset Statistics

Table A.1 summarises dataset size, class balance, temporal span, and account activity. The datasets range from about five million to 180 million transactions and from about 0.5 million to 2.1 million accounts. In contrast, the median number of counterparties grows from two to five. Thus, the scale increase is driven mainly by more transactions, accounts, and longer temporal coverage rather than denser local structure.

Table A.1: Size, class balance, temporal span and aggregated degree of the six AMLWorld variants. *Ill.%* is the proportion of illicit transactions. *Degree* counts distinct counterparties per account.

Dataset	#Accounts	#Transactions	#Illicit	Ill.%	Span (days)	Degree	
						Mean	Median
HI-Small	515,088	5,078,345	5,177	0.102	10	3.2	2
LI-Small	705,907	6,924,049	3,565	0.051	10	3.2	2
HI-Medium	2,077,023	31,898,238	35,230	0.110	16	3.6	3
LI-Medium	2,032,095	31,251,483	16,041	0.051	16	3.6	3
HI-Large	2,116,168	179,702,229	225,546	0.126	97	6.8	5
LI-Large	2,070,980	176,066,557	100,604	0.057	97	6.7	5

AMLWorld records the amount and the currency on both the sending and the receiving side of each transaction. Our models use the receiver-side fields, Amount Received and Receiving Currency, following the Multi-GNN and MEGA-GNN baselines. The two sides disagree on 1.4 to 1.6 percent of transactions across all variants (Table A.2), none of which are labelled as illicit.

Table A.2: Where the sender-side and receiver-side transaction fields differ. The two conditions coincide except for a few transactions per dataset where only the currency differs, none has only the amount differ, and none is labelled illicit.

	HI-S	LI-S	HI-M	LI-M	HI-L	LI-L
Currency or amount differs, % of transactions	1.421	1.428	1.521	1.497	1.557	1.532
Currency and amount both differ, % of transactions	1.421	1.428	1.521	1.497	1.556	1.532
Currency differs but amount matches, count	12	18	57	73	281	380

Table A.3 reports the composition of each split. The evaluation splits are more imbalanced than the training splits. Not all datasets follow a 60/20/20 transaction split, because the temporal splits are defined by day boundaries. The same boundaries are used for all evaluated models, and follow the protocol used by the Multi-GNN and MEGA-GNN baselines.

Table A.3: Composition of the temporal splits. Transactions, share of the dataset, and illicit rate within the split.

Dataset	Train			Validation			Test		
	#Txns	Share	Ill.%	#Txns	Share	Ill.%	#Txns	Share	Ill.%
HI-Small	3.25M	64%	0.078	966k	19%	0.107	864k	17%	0.186
LI-Small	4.43M	64%	0.045	1.32M	19%	0.058	1.17M	17%	0.068
HI-Medium	19.49M	61%	0.081	5.52M	17%	0.154	6.89M	22%	0.160
LI-Medium	19.10M	61%	0.043	5.41M	17%	0.070	6.74M	22%	0.059
HI-Large	108.29M	60%	0.100	35.32M	20%	0.129	36.09M	20%	0.198
LI-Large	106.14M	60%	0.052	34.60M	20%	0.060	35.33M	20%	0.071

Table A.4 analyses the account history lengths. Only about 15 accounts per dataset exceed Chronos-2’s encoder context of 8192 transactions. However, truncation removes 3.3 to 4.8 percent of all history transactions. The effect is concentrated in a small set of highly active accounts rather than spread uniformly across the data. The last column reports the same share computed over illicit transactions only. On the HI variants it stays close to the overall share, but on the LI variants it is two to three times higher, reaching 12 percent on LI-Medium, so truncation does fall more heavily on the labelled transactions there. Those transactions still sit in the histories of the same few hub accounts, and it is the oldest part of each history that is dropped.

Table A.4: Account history lengths against the 8192 transaction encoder context. L_a counts transactions incident to an account, here the full history at the test cutoff. *Transactions cut* is the share of history transactions lost to truncation, given over all transactions and over illicit transactions only.

Dataset	History length L_a				Over 8192		Transactions cut %	
	Median	Mean	P99	Max	#Acc.	%	All	Illicit
HI-Small	6	19.7	119	169,756	14	0.0027	3.26	4.73
LI-Small	6	19.6	119	223,590	13	0.0018	3.55	9.34
HI-Medium	7	30.7	193	1,083,821	15	0.0007	4.48	5.68
LI-Medium	7	30.8	194	1,099,681	15	0.0007	4.45	12.24
HI-Large	33	169.8	1118	6,265,868	15	0.0007	4.76	5.36
LI-Large	33	170.0	1121	6,359,429	15	0.0007	4.74	11.67

A.2. Laundering Typologies

AMLWorld documents a subset of its laundering as patterns. Only transactions listed in the pattern files provided with the datasets on Kaggle¹ carry a typology, and those files cover 61 to 65 percent of the labelled laundering on the HI variants and only 19 to 29 percent on the LI variants (last row of Table A.5), so what follows describes the structure AMLWorld chose to document rather than the whole label set.

The typologies split by how much of a pattern one account can see. Fan-in, fan-out and gather-scatter are fully local, one account is party to every transaction, so an account can see the whole pattern in its own history. These patterns run about 9 to 16 transactions. Scatter-gather is half visible from one account, cycle and random about a third, and stack and bipartite between 13 and 21 percent, so those patterns are spread across accounts and message passing could help reach them.

A.3. Pair Structure

The pair extension aggregates transactions between two accounts. Table A.6 summarises the distribution of the transactions in the pairs.

¹<https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-money-laundering-aml>

Table A.5: How much of a laundering pattern one account can see. Each cell is the mean, over the typology’s patterns of at least three transactions, of the largest share of a pattern’s transactions that any single account is party to. *Txns* is the median size of the counted patterns and *Share of laundering* the fraction of the variant’s labelled laundering the pattern files document.

Typology	% of a pattern one account is party to						Txns
	HI-S	LI-S	HI-M	LI-M	HI-L	LI-L	
Fan-in	100	100	100	100	100	100	9
Fan-out	100	100	100	100	100	100	9
Gather-scatter	100	100	100	100	100	100	16
Scatter-gather	53	50	57	54	56	56	16
Random	32	34	33	34	33	34	7
Cycle	39	35	32	33	34	32	7
Stack	17	20	20	21	18	19	16
Bipartite	15	13	14	13	15	15	9
Share of laundering %	62	29	65	24	61	19	

Table A.6: Pair multiset sizes over the whole dataset. $|M_{uv}|$ counts the transactions between an unordered account pair in either direction. *Singleton* gives the share of pairs holding exactly one transaction, and the share of transactions and of illicit transactions sitting in such a pair. *One way* is the share of pairs whose transactions all run in the same direction, over all pairs and over pairs of at least two transactions only, since a singleton pair is one way by construction. The last two columns price the 64 transaction cap, only TCEE-PNA + Pair-PNA and TCEE-PNA + Pair-PNA + Nbr-Attn models on the Large datasets.

Dataset	Pairs	$ M_{uv} $				Singleton %			One way %		Over 64 %	
		Median	Mean	P99	Max	Pairs	Txns	Illicit	All	≥ 2	Pairs	Transactions
HI-Small	1,012,043	2	5.02	30	89	44.2	8.8	69.5	99.6	99.3	0.00	0.00
LI-Small	1,380,011	2	5.02	30	81	44.1	8.8	55.9	99.6	99.4	0.00	0.00
HI-Medium	4,443,981	2	7.18	48	118	46.0	6.4	63.9	99.3	98.6	0.12	0.25
LI-Medium	4,331,289	2	7.22	48	145	45.9	6.4	47.9	99.3	98.6	0.12	0.25
HI-Large	7,967,550	2	22.55	291	690	40.8	1.8	50.4	93.7	89.4	10.77	51.67
LI-Large	7,689,013	3	22.90	291	877	40.1	1.8	22.3	93.6	89.4	10.95	51.76

Pair histories are much shorter for illicit transactions overall. Across the six variants, an illicit transaction is 6 to 28 times more likely than an arbitrary one to be the only transaction in its pair. The effect varies by dataset however.

For singleton pairs, PNA aggregation contributes nothing beyond the transaction itself, since the mean, minimum and maximum coincide and the standard deviation is zero. On those edges the pair extension can only add the pair degree. This is a property of the aggregator rather than a measurement, so it bounds what the extension can contribute on singleton edges.

Almost every pair is unidirectional, and this holds even among the pairs that carry several transactions, so it is not an artefact of the many singleton pairs. This could explain why making the pair aggregation explicitly directional does not provide a useful additional signal in the ablation of Section B.3.

Pair size and memory. In our implementation, the pair gather pads every pair in a batch to the batch’s longest, so its cost follows the maximum. On the Small and Medium variants the longest pairs are short enough for this to cost little. The Large variants hold pairs of several hundred transactions however, and the padded activation no longer fits on the GPU beside the account embedding store, so the TCEE-PNA runs carrying the pair extension cap the gather at 64 transactions.

A.4. Neighbourhood Window Size

The neighbourhood extension reads a window of $2w$ transactions from each of the two accounts a transaction runs between, so it reads $\min(L_u, 2w) + \min(L_v, 2w)$ transactions, $4w$ at most. Table A.7 reports the median count a scored transaction reads at each candidate w .

Table A.7: Median transactions the neighbourhood window reads per scored transaction against the half-window w , over the whole dataset, with that median as a share of the $4w$ budget in parentheses and shaded by that share. *Median L_a* is the median history length of a transaction’s sender, its receiver, and both together.

Half-window w							Median L_a		
	2	4	8	16	32	64	Sender	Receiver	All
HI-Small	8 (100)	16 (100)	32 (100)	64 (100)	91 (71)	107 (42)	63	36	46
LI-Small	8 (100)	16 (100)	32 (100)	64 (100)	91 (71)	107 (42)	63	36	46
HI-Medium	8 (100)	16 (100)	32 (100)	64 (100)	116 (91)	165 (64)	106	59	76
LI-Medium	8 (100)	16 (100)	32 (100)	64 (100)	117 (91)	165 (64)	106	59	76
HI-Large	8 (100)	16 (100)	32 (100)	64 (100)	128 (100)	256 (100)	632	344	450
LI-Large	8 (100)	16 (100)	32 (100)	64 (100)	128 (100)	256 (100)	632	345	450

Up to $w = 16$ the median scored transaction reads a full window on every variant. Past that the Small datasets run out of history, so a $w = 32$ window is 71% used and a $w = 64$ one only 42%, while the Medium datasets hold up better at 91% and 64% and the Large datasets stay full throughout.

A.5. Experimental Details

Table A.8 gives the experimental details shared by all reported runs.

The PNA backbone on the Large variants completes four epochs plain and one to two with the pair extension or with both extensions, against budgets of 20 to 50 epochs for the MLP models on the same data. A single epoch there costs about 19 hours plain and about 140 hours with both extensions. Those scores should therefore be read as lower bounds rather than as what the architecture reaches when trained to convergence. The MLP variants carry the best reported Large results likely for this reason.

Table A.8: Experimental details shared by all reported runs.

Setting	Value
Optimiser	Adam, weight decay 0
Learning rate	3×10^{-4} TCEE encoder, 3×10^{-3} Chronos pool
	6.1×10^{-4} PNA backbone, 3.2×10^{-3} GIN backbone, 10^{-3} MLP heads
Loss	Class weighted cross entropy, minority weight 6.275
Batch size	8192 edges
Neighbour sampling	[100, 100] for message passing backbones
Epochs	See Table A.9
Projection width d_{proj}	16, Chronos models
Attention width d_{attn}	32, TCEE models
Neighbourhood half-window w	16
Encoder context	8192 transactions per account, frozen Chronos-2
Seeds	42, 43, 44, 45, 46
Checkpoint selection	Epoch with the best validation value of the reported metric
Hardware	One NVIDIA A40

Table A.9: Epoch budget / best validation PR-AUC epoch, the latter as a minimum to maximum over the five seeds. * marks the arms cut by the wall-clock limit before reaching their budget.

Model	HI-Small	LI-Small	HI-Medium	LI-Medium	HI-Large	LI-Large
Chronos-MLP	50 / 4–11	50 / 3–10	50 / 10–22	50 / 5–14	50 / 18–32	50 / 14–33
Chronos-PNA	50 / 10–20	50 / 12–24	20 / 16–19	20 / 14–20	6 / 4–6	6 / 5–6
TCEE-MLP	50 / 6–15	50 / 4–17	50 / 12–19	50 / 5–18	50 / 16–28	50 / 13–26
+ Pair-PNA	50 / 7–15	50 / 6–15	50 / 13–23	50 / 14–25	20 / 10–19	20 / 12–20
+ both	50 / 11–26	50 / 6–15	50 / 15–18	50 / 12–19	20 / 9–18	20 / 12–20
TCEE-PNA	50 / 6–18	50 / 8–20	20 / 14–18	20 / 13–20	4 / 4	4 / 3–4
+ Pair-PNA	50 / 13–20	50 / 18–27	20 / 10–20	20 / 13–20	3 / 1*	3 / 1–2*
+ both	50 / 15–21	50 / 8–16	20 / 12–17	20 / 15–18	3 / 1*	3 / 1*

B

Additional Ablations

This chapter examines the main design choices behind TCEE. We consider the role of the frozen context and its pretraining, the input features, the pair and neighbourhood extensions, and the sensitivity of the architecture to its main width. Unless stated otherwise, reported test PR-AUC values are means with sample standard deviations over five seeds.

B.1. Contribution of Chronos and Pretraining

The frozen Chronos encoder provides a representation of each account’s transaction history and the temporal information learned during pretraining. We first measure the effect of including this context after adding the extensions (Section B.1.1), and then separate the effect of the encoder architecture from that of pretraining (Section B.1.2).

B.1.1. Chronos Contribution

Removing the Chronos embeddings leaves no context to attend over and reduces the model to an MLP over the transaction representation and any active extension.

Table B.1: TCEE-MLP with and without the frozen Chronos context.

Variant	Context	HI-S	LI-S	HI-M	LI-M	HI-L	LI-L
TCEE-MLP	Without	55.01 $\pm$ 0.87	23.63 $\pm$ 0.81	60.99 $\pm$ 0.23	21.27 $\pm$ 0.25	66.80 $\pm$ 0.19	23.30 $\pm$ 0.30
	With	69.24 $\pm$ 0.61	31.45 $\pm$ 0.93	74.70 $\pm$ 0.53	35.25 $\pm$ 0.36	75.50 $\pm$ 0.96	32.67 $\pm$ 0.77
+ Pair-PNA	Without	69.97 $\pm$ 0.87	32.04 $\pm$ 1.40	75.83 $\pm$ 0.10	35.46 $\pm$ 0.60	77.96 $\pm$ 0.13	34.93 $\pm$ 0.17
	With	75.00 $\pm$ 0.60	34.04 $\pm$ 1.15	79.11 $\pm$ 0.10	37.98 $\pm$ 0.91	80.31 $\pm$ 0.37	38.18 $\pm$ 0.54
+ Nbr-Attn	Without	75.54 $\pm$ 1.33	36.10 $\pm$ 1.48	78.51 $\pm$ 0.35	38.94 $\pm$ 0.39	80.41 $\pm$ 0.15	37.23 $\pm$ 0.49
	With	75.51 $\pm$ 0.87	34.99 $\pm$ 1.28	78.97 $\pm$ 0.31	38.42 $\pm$ 0.87	80.26 $\pm$ 0.07	38.48 $\pm$ 0.18
+ both	Without	75.38 $\pm$ 2.35	31.50 $\pm$ 1.32	80.94 $\pm$ 0.10	41.76 $\pm$ 0.23	81.55 $\pm$ 0.07	39.68 $\pm$ 0.32
	With	78.26 $\pm$ 1.13	34.06 $\pm$ 1.05	81.00 $\pm$ 0.03	41.12 $\pm$ 0.43	81.61 $\pm$ 0.09	40.39 $\pm$ 0.11

Table B.1 compares each configuration with and without the frozen context. Without extensions, adding the context increases test PR-AUC by 33.4% on average over all datasets. Thus, in this setting, the frozen context provides a substantial part of the predictive signal. Once both extensions are present however, adding the frozen context shows minimal benefit. On the Medium and Large datasets the differences are within a point in either direction, while on the Small datasets the context is still worth about two and a half points although with a large variance.

Chronos-2 still consistently improves PR-AUC with the Pair-PNA extension enabled, but when the neighbourhood extension is introduced, its contribution becomes largely redundant and at times harmful (on LI-Small). While this answers how the frozen context contributes to the final TCEE-MLP model, the effects might not generalise to TCEE-PNA or other architectures. Further experiments would be needed to determine the role of the frozen context in those settings.

B.1.2. Pretraining Contribution

Table B.2: Plain TCEE-MLP on embeddings re-extracted with a randomly initialised Chronos-2.

Context source	HI-S	LI-S	HI-M	LI-M	HI-L	LI-L
No context	55.01 $\pm$ 0.9	23.63 $\pm$ 0.8	60.99 $\pm$ 0.2	21.27 $\pm$ 0.2	66.80 $\pm$ 0.2	23.30 $\pm$ 0.3
Random-init encoder	63.61 $\pm$ 1.7	23.44 $\pm$ 2.4	74.38 $\pm$ 0.4	34.70 $\pm$ 0.4	76.48 $\pm$ 0.4	33.22 $\pm$ 0.5
Pretrained (default)	69.24 $\pm$ 0.6	31.45 $\pm$ 0.9	74.70 $\pm$ 0.5	35.25 $\pm$ 0.4	75.50 $\pm$ 1.0	32.67 $\pm$ 0.8

To separate architecture from pretraining, we re-extract the store with a randomly initialised Chronos-2 of the same architecture (Table B.2). The random encoder performs substantially better than removing the encoder (No context) on most datasets, showing that the encoder architecture itself provides useful representations. On LI-Small, however, the random encoder performs no better than removing the context. The gap between the random and pretrained encoders is largest on the Small datasets and small on the Medium and Large datasets. This suggests that pretraining may be more useful when labelled data are limited. Testing domain-specific pretraining should be explored in future work as the encoder used here was not pretrained on anti-money laundering data.

B.2. Information Sources

The Chronos input combines the transaction fields with two additional signals, the time since the account's previous transaction and a counterparty index. We examine the contribution of these signals and whether their particular encoding matters.

B.2.1. Effect of Time and Counterparty Information

Table B.3 starts from the full feature set and removes the time delta, the counterparty index, or both. We evaluate the resulting representations with Chronos-MLP, TCEE-MLP, and Edge-MLP, which is a pure MLP over TCEE-MLP's transaction features with no frozen encoder. The None row still keeps the amount, currency, payment type and incoming flag.

Table B.3: Time delta and counterparty index removed from the full model inputs.

Input	HI-Small			LI-Small		
	Edge-MLP	Chronos-MLP	TCEE-MLP	Edge-MLP	Chronos-MLP	TCEE-MLP
All features	16.89 $\pm$ 2.73	58.92 $\pm$ 3.39	69.24 $\pm$ 0.61	1.60 $\pm$ 0.32	20.16 $\pm$ 3.53	31.45 $\pm$ 0.93
– Time delta	8.38 $\pm$ 1.11	58.93 $\pm$ 4.26	64.83 $\pm$ 1.47	3.09 $\pm$ 1.78	22.71 $\pm$ 2.42	29.94 $\pm$ 1.38
– Counterparty index	10.54 $\pm$ 1.15	54.69 $\pm$ 4.15	63.20 $\pm$ 2.84	1.45 $\pm$ 0.10	18.80 $\pm$ 3.16	27.64 $\pm$ 0.83
None	1.74 $\pm$ 0.49	55.23 $\pm$ 1.02	59.07 $\pm$ 1.82	0.78 $\pm$ 0.44	16.11 $\pm$ 4.04	23.74 $\pm$ 2.00

Table B.3 shows that both signals contribute to TCEE-MLP. Removing the counterparty index costs 6.0 test points on HI-Small and 3.8 on LI-Small, removing the time delta costs 4.4 and 1.5, and removing both costs 10.2 and 7.7. The counterparty index matters more than the time delta on both datasets.

The lower capacity models are less consistent. For Chronos-MLP, removing the time delta alone leaves HI-Small unchanged and raises LI-Small by about two and a half points, while removing the counterparty index still costs a few points. Edge-MLP stays below 17 test PR-AUC throughout and falls to near zero once both signals are gone, so without the frozen context neither signal carries the task on its own.

B.2.2. Representing Time and Counterparty

We next compare alternative encodings for the two signals, time in Table B.4 and the counterparty in Table B.5. Time is represented either by the absolute hour offset or by the gap since the account's previous transaction. Counterparty information is represented either by the full counterparty index or by a binary flag that is set when the account has not transacted with that counterparty before.

For TCEE-MLP, time delta gives higher test PR-AUC than either absolute offset on both Small datasets. We select it because it also has the highest average validation PR-AUC across the two datasets on TCEE-MLP. The time delta is also a more stable quantity across temporal splits, an offset measured from the first transaction grows over time, whereas the gap to the previous transaction does not have the same cumulative drift.

Table B.4: Temporal encoding, absolute offset against time delta, for TCEE-MLP without extensions and Chronos-MLP. Counterparty information is excluded here. Each cell is validation / test PR-AUC. The highest validation score in each column is in bold.

Setting	TCEE-MLP		Chronos-MLP	
	HI-Small	LI-Small	HI-Small	LI-Small
Hour offset	49.4 $\pm$ 0.7 / 59.5 $\pm$ 2.4	18.7 $\pm$ 0.4 / 24.1 $\pm$ 1.3	51.9 $\pm$ 0.2 / 58.8 $\pm$ 2.0	18.1 $\pm$ 0.6 / 19.2 $\pm$ 1.8
Minute offset	49.3 $\pm$ 1.3 / 58.5 $\pm$ 0.9	18.8 $\pm$ 0.6 / 24.4 $\pm$ 1.3	51.5 $\pm$ 0.3 / 57.7 $\pm$ 3.1	17.9 $\pm$ 0.6 / 19.1 $\pm$ 2.2
Time delta	51.3 $\pm$ 0.5 / 63.2 $\pm$ 2.8	18.3 $\pm$ 0.5 / 27.6 $\pm$ 0.8	51.8 $\pm$ 0.7 / 54.7 $\pm$ 4.1	18.1 $\pm$ 0.6 / 18.8 $\pm$ 3.2

Table B.5: Counterparty encoding, flag against index. *Flag* marks a first-time counterparty, *index* is the running counterparty identifier. Each cell is validation / test PR-AUC. The higher validation score of each pair is in bold.

Configuration	HI-Small		LI-Small	
	Flag	Index	Flag	Index
<i>Without extensions</i>				
Chronos-MLP	53.4 $\pm$ 0.8 / 62.1 $\pm$ 1.8	52.3 $\pm$ 0.8 / 62.8 $\pm$ 1.7	20.5 $\pm$ 0.9 / 27.1 $\pm$ 2.4	18.3 $\pm$ 0.7 / 25.8 $\pm$ 1.8
Chronos-PNA	57.2 $\pm$ 0.8 / 65.0 $\pm$ 2.0	56.1 $\pm$ 1.1 / 64.9 $\pm$ 2.2	21.9 $\pm$ 0.5 / 24.5 $\pm$ 1.9	20.5 $\pm$ 1.5 / 24.9 $\pm$ 3.3
TCEE-MLP	57.6 $\pm$ 0.6 / 70.0 $\pm$ 0.7	56.5 $\pm$ 1.0 / 69.2 $\pm$ 0.6	23.6 $\pm$ 0.5 / 34.1 $\pm$ 1.0	22.8 $\pm$ 0.4 / 31.4 $\pm$ 0.9
TCEE-PNA	59.4 $\pm$ 1.3 / 73.0 $\pm$ 1.5	59.3 $\pm$ 0.6 / 74.3 $\pm$ 1.3	24.9 $\pm$ 1.4 / 35.4 $\pm$ 1.5	24.2 $\pm$ 1.0 / 31.5 $\pm$ 2.2
<i>With extensions</i>				
TCEE-MLP + Pair-PNA	60.3 $\pm$ 0.9 / 73.0 $\pm$ 1.8	60.9 $\pm$ 0.6 / 75.0 $\pm$ 0.6	25.4 $\pm$ 0.5 / 33.5 $\pm$ 0.9	26.5 $\pm$ 0.4 / 34.0 $\pm$ 1.1
TCEE-PNA + Pair-PNA	63.6 $\pm$ 0.1 / 78.3 $\pm$ 0.8	63.6 $\pm$ 0.5 / 79.0 $\pm$ 0.7	27.8 $\pm$ 0.9 / 37.0 $\pm$ 0.9	27.9 $\pm$ 0.4 / 36.1 $\pm$ 0.8
TCEE-MLP + Pair-PNA + Nbr-Attn	63.0 $\pm$ 0.8 / 76.3 $\pm$ 0.7	64.0 $\pm$ 0.6 / 78.3 $\pm$ 1.1	26.4 $\pm$ 0.5 / 34.2 $\pm$ 0.8	27.2 $\pm$ 0.7 / 34.1 $\pm$ 1.0
TCEE-PNA + Pair-PNA + Nbr-Attn	64.5 $\pm$ 1.1 / 78.7 $\pm$ 0.9	64.6 $\pm$ 0.6 / 79.9 $\pm$ 0.8	27.2 $\pm$ 0.9 / 36.2 $\pm$ 1.3	28.8 $\pm$ 0.7 / 37.3 $\pm$ 1.5

The flag and index encodings are broadly similar. Without extensions, the flag is higher on validation for every configuration. With extensions the index is higher in seven of the eight, and equal on TCEE-PNA + Pair-PNA on HI-Small. We therefore retain the index, mainly because it gives the better validation score for the with-extensions models and retains more information than a binary flag.

B.2.3. Account Views of the Transaction

Table B.6: Chronos-MLP with the sender and receiver views of the transaction added to its classifier input.

Model	HI-S	LI-S	HI-M	LI-M	HI-L	LI-L
Chronos-MLP	58.92 $\pm$ 3.4	20.16 $\pm$ 3.5	65.15 $\pm$ 1.0	27.30 $\pm$ 1.4	64.89 $\pm$ 1.1	23.79 $\pm$ 0.8
Chronos-MLP + per-account views	66.14 $\pm$ 1.4	29.65 $\pm$ 2.1	71.48 $\pm$ 0.6	31.64 $\pm$ 1.0	72.37 $\pm$ 0.9	29.27 $\pm$ 0.8
TCEE-MLP	69.24 $\pm$ 0.6	31.45 $\pm$ 0.9	74.70 $\pm$ 0.5	35.25 $\pm$ 0.4	75.50 $\pm$ 1.0	32.67 $\pm$ 0.8

Chronos-MLP and TCEE-MLP read the same frozen embeddings but describe the scored transaction differently. Chronos-MLP, created as a baseline, passes its classifier only the four raw transaction fields, while TCEE-MLP builds a sender view and a receiver view of the transaction from all six input series and uses them as its queries. To separate this richer description from the retrieval itself, we add both views to the classifier input of Chronos-MLP. Table B.6 shows that the account views recover between 55 and 84 percent of the gap between Chronos-MLP and TCEE-MLP across all datasets. Most of the improvement of plain TCEE-MLP over Chronos-MLP therefore likely comes from how the transaction is represented, while the edge-conditioned retrieval adds a smaller part (Section B.5.3).

B.3. Pair Context Design

The pair extension summarises the transactions exchanged between the two accounts involved in the scored transaction. We compare PNA pooling with directional pooling (pooling with PNA separately for each direction) and with attention over the pair’s transactions. We also include Chronos-Edge-MLP, which encodes the sequence of parallel transactions with Chronos-2 and combines the result with the transaction features through an MLP.

The three direct aggregation mechanisms perform similarly, and neither directional pooling nor attention consistently improves on PNA pooling. We therefore retain PNA as the default. The directional variant also adds complexity when most pairs contain transactions in only one direction (Section A.3). Chronos-

Table B.7: Pair extension design for TCEE-MLP.

Pair mechanism	HI-Small	LI-Small
Pair pooling with PNA (default)	75.00 $\pm$ 0.60	34.04 $\pm$ 1.15
Directional pair pooling	73.79 $\pm$ 1.21	35.09 $\pm$ 0.80
Attention over the pair's transactions	73.76 $\pm$ 0.39	34.15 $\pm$ 1.12
Chronos-Edge-MLP	32.70 $\pm$ 5.20	4.50 $\pm$ 0.50

Edge-MLP performs much worse in this comparison, especially on LI-Small. Pair histories are short with the median being two transactions, and 44% of pairs contain only one transaction on the Small datasets. The statistics are similar for the rest of the datasets. In this setting, direct aggregation appears better suited to the pair context than applying the frozen sequence encoder.

B.4. Neighbourhood Window

The neighbourhood extension attends to a window of $2w$ transactions around the scored transaction in each account's own history, the w before it and the w from it onward. We vary w from 1 to 64 on the Small and Medium datasets, keeping the rest of the model fixed, and ask how much the selected default of $w=16$ matters.

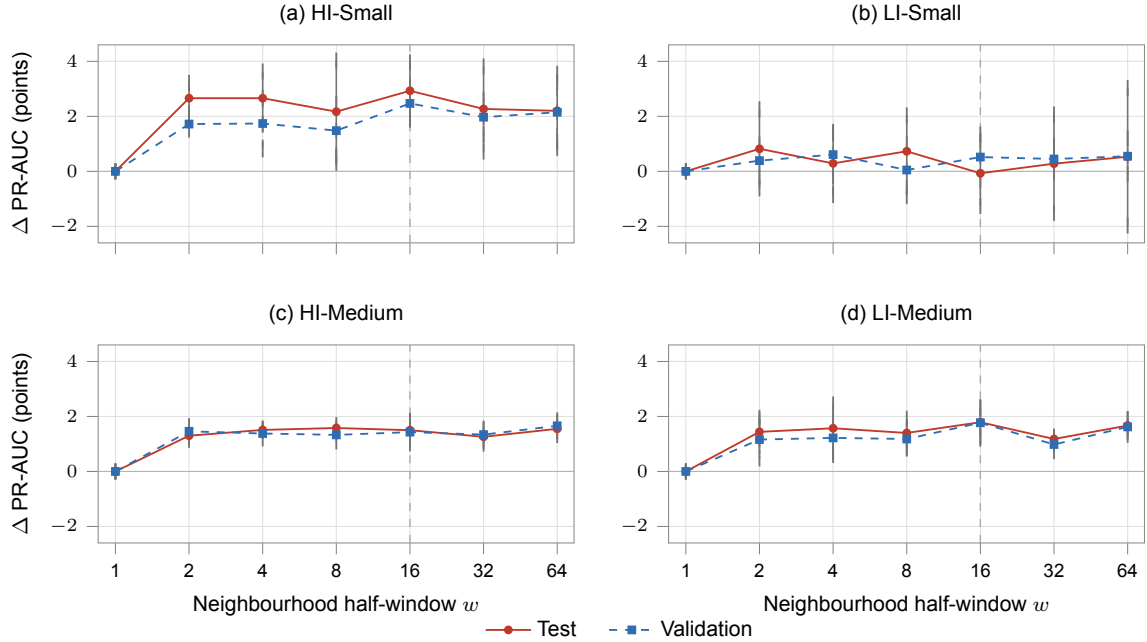


Figure B.1: The full TCEE-MLP against the neighbourhood half-window w , so the window holds $2w$ transactions per account. Each series is shown relative to its own PR-AUC at $w=1$, with standard deviation bars on the paired per-seed difference. The dashed line marks the selected default $w=16$. Small datasets are five seeds, Medium three.

Figure B.1 shows each setting relative to its own performance at $w=1$. On three of the four datasets, performance increases once w reaches 2, a window of four transactions, and then remains largely stable. LI-Small is largely insensitive to the window size which is consistent with the earlier findings that Nbr-Attn adds little for this dataset once Pair-PNA is present. The sharpest change therefore occurs between $w=1$ and $w=2$, after which additional context provides little consistent benefit.

Comparing with the local structure of the datasets (Table A.5), the window of four transactions is enough to mostly cover entire local patterns of fan-in, fan-out where the median pattern is nine transactions long and a single account is party to all of them, meaning they appear all in the same window. This is not the case for all patterns however. In a cycle a single account is party to only 32 to 39% of the pattern.

Larger windows do not consistently improve performance, including on the Medium datasets. This

could be because the model might only need to see part of the pattern to detect it. On the Small datasets this may also reflect the shorter account histories, which limit the additional context available at larger windows (Section A.4). The exact reason should be further investigated.

The default $w=16$ was selected using validation PR-AUC averaged over the two Small datasets, although the sweep suggests that the exact choice of w has limited effect on the reported results past $w=2$.

B.5. Architectural Sensitivity

We examine how sensitive TCEE and its baselines are to the two width choices and to the parameter sharing in the attention core. Each was fixed on validation PR-AUC over the two Small datasets, and each comparison below asks how much that choice affected the results.

B.5.1. Projection Width

Models that pool the frozen embeddings first project them from 768 dimensions to d_{proj} . We sweep this width from 4 to the uncompressed 768 for Chronos-MLP under each of the three feature poolings, attention, mean and concatenation.

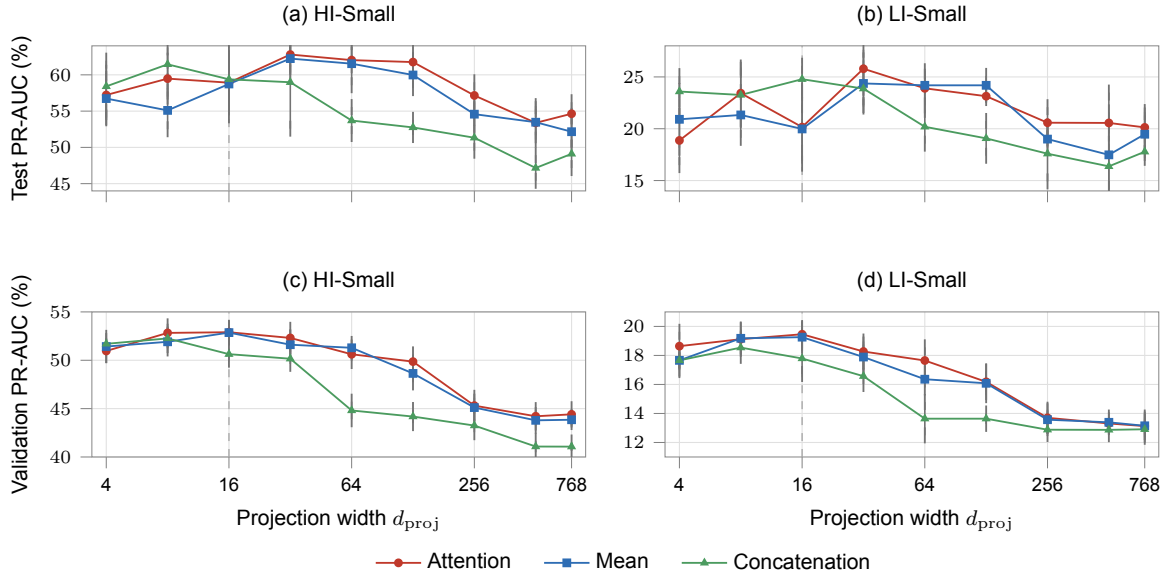


Figure B.2: Chronos-MLP against the projection width d_{proj} , for the three feature poolings. The dashed line marks the default $d_{\text{proj}}=16$.

Figure B.2 shows a broad performance plateau at moderate projection widths for all three pooling methods, followed by lower scores at wider projections. One possible explanation is that wider projections increase the number of trainable parameters while the labelled data are unchanged. They may also retain dimensions that are not useful for the downstream task. At small widths, concatenation can perform well, but its performance declines more sharply as the projection widens.

We select $d_{\text{proj}}=16$ with attention pooling because it gives the highest average validation PR-AUC across the two Small datasets. At this width, attention and mean pooling are very close, while attention is ahead of concatenation on validation for every seed.

B.5.2. Attention Width

We examine how sensitive TCEE is to the width of the representation it uses to read the frozen store. Since $\mathbf{W}_K$ and $\mathbf{W}_V$ project the 768 dimensional embeddings to d_{attn} , this width sets the capacity of the retrieval mechanism. We sweep it from 4 to 256 on the plain TCEE-MLP and also probe $d_{\text{attn}}=1$ and 2. At each width we compare the query-weighted attention over the feature vectors against a plain mean over them.

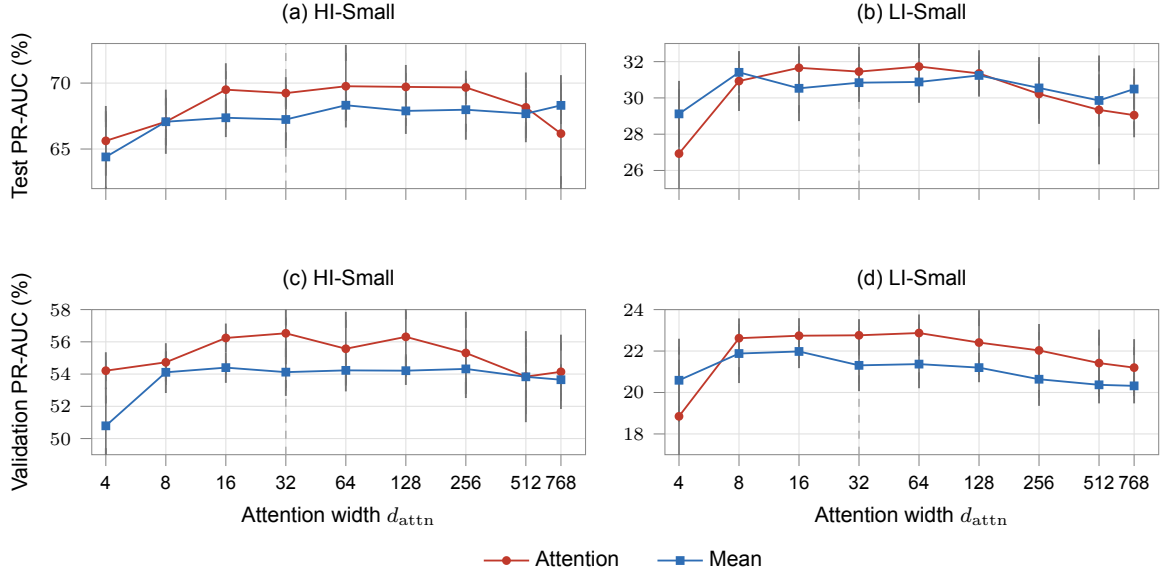


Figure B.3: The plain TCEE-MLP against the attention width d_{attn} . The dashed line marks the default $d_{\text{attn}}=32$.

Figure B.3 shows that the plain TCEE-MLP is insensitive to the attention width for the Small datasets once d_{attn} reaches about 8. Very small widths are unstable, at $d_{\text{attn}}=2$ test PR-AUC falls to 29.3 on HI-Small and to about 1 on LI-Small, and it is approximately zero at $d_{\text{attn}}=1$. These two points are left out of the figure, since they would compress the rest of the curve.

The figure also plots the same model with the attention over the feature axis replaced by a plain mean. The attention generally stays at or above the mean on validation, the exception being $d_{\text{attn}}=4$ on LI-Small, and any small gains it gives over the mean (Section B.5.3) do not seem to depend on the attention width.

The width itself has little effect once it is not too small, while there seems to be a small downwards trend as d_{attn} increases, it is not as evident as in d_{proj} (Section B.5.1).

B.5.3. Edge-Conditioned Retrieval

The retrieval in TCEE attends over each account's per-feature embeddings with the scored transaction as the query. This subsection replaces that attention with a plain mean over the feature axis F , keeping the patch axis meaned and every other setting fixed.

Table B.8: Mean against attention over the feature axis F , for Chronos-MLP and TCEE-MLP.

Model	Read over F	HI-S	LI-S	HI-M	LI-M	HI-L	LI-L
Chronos-MLP	Mean	58.74 $\pm$ 4.3	19.98 $\pm$ 2.3	65.95 $\pm$ 1.4	27.49 $\pm$ 1.3	64.34 $\pm$ 0.4	22.94 $\pm$ 1.1
	Attention	58.92 $\pm$ 3.4	20.16 $\pm$ 3.5	65.15 $\pm$ 1.0	27.30 $\pm$ 1.4	64.89 $\pm$ 1.1	23.79 $\pm$ 0.8
TCEE-MLP	Mean	67.24 $\pm$ 1.6	30.84 $\pm$ 0.6	73.20 $\pm$ 0.4	33.62 $\pm$ 1.1	74.21 $\pm$ 1.2	31.99 $\pm$ 0.7
	Attention	69.24 $\pm$ 0.6	31.45 $\pm$ 0.9	74.70 $\pm$ 0.5	35.25 $\pm$ 0.4	75.50 $\pm$ 1.0	32.67 $\pm$ 0.8

Table B.8 shows that for Chronos-MLP the mean and the attention over F are within seed variation of each other on every dataset. This is the setting where the query carries no transaction-specific information, so there is nothing for the attention to weight the features with, and the table is consistent with it collapsing to a mean.

For TCEE-MLP the attention is above the mean on every dataset, by 2.0 and 0.6 test points on the Small datasets, 1.5 and 1.6 on the Medium datasets, and 1.3 and 0.7 on the Large datasets. Not all differences are within seed variation and the attention is consistently above the mean, suggesting that the edge-conditioned retrieval can give a small benefit. However, the effect is not large enough to explain the large gains of TCEE-MLP.

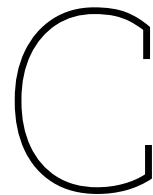
B.5.4. Parameter Sharing

TCEE shares one key and value projection across all accounts and series, and one query MLP across both account views. We compare this with separate query MLPs for the two account roles and with separate key projections for each series.

Table B.9: Sensitivity to parameter sharing in the attention core of TCEE-MLP, with both extensions active. Each variant changes only the indicated projection.

Attention core	HI-Small	LI-Small
Shared keys, shared query MLP (default)	78.26 $\pm$ 1.13	34.06 $\pm$ 1.05
Separate query MLP per account	77.23 $\pm$ 0.87	34.44 $\pm$ 0.71
Separate key projection per feature	77.18 $\pm$ 0.97	35.63 $\pm$ 1.22

Table B.9 shows no consistent benefit from unsharing. The alternatives are about one point below the default on HI-Small and close to or above it on LI-Small. We retain the shared configuration because it uses fewer parameters and performs similarly on average. Together with the width sweeps, these results suggest that a higher number of parameters does not necessarily improve performance, although whether this also holds on larger datasets is not measured here and should be further investigated.



Attention Analysis

The model exposes attention weights over each account's six feature series and over the transactions in each account's window. In this appendix we measure how selective the weights are, whether they focus on prediction-relevant information, and whether they can help an analyst in the review process.

C.1. Measurement Protocol

The experiments in this section use the following protocol.

Checkpoints

The concentration measurements (Sections C.2.1 and C.2.2) read the full TCEE-MLP on HI-Small, the variant with a window of surrounding transactions, since they describe the model reported elsewhere in the thesis. The feature-wise weights (Section C.2.3) and the importance tests (Sections C.3.1 and C.3.2) read the plain TCEE-MLP instead, because in the full model a feature can also reach the prediction through the pair and neighbourhood branches, so a low attention weight there would not mean the feature is unused, and because the feature-wise weights are later compared against a plain-model importance ordering. Section C.3.3 returns to the full model, since the plain one has no window to delete from. The parts on different checkpoints do not describe exactly the same attention weights.

Sampling

Each of the five checkpoints (seeds 42–46) is run once on the test split. Each transaction contributes two rows, one for each account view (receiver/sender), and transactions are sampled uniformly within class. The attention pass includes all 1,611 illicit transactions and 5,000 of the 862,289 licit transactions, giving 3,222 and 10,000 account views. The feature deletion pass is balanced instead, using all 1,611 illicit transactions and 1,611 licit ones. Reported values are means of the five per-seed values, with their standard deviations.

C.2. Attention Selectivity

The thesis claims a retrieval conditioned on the individual transaction, however the weights could be nearly flat, in which case an average would do the same work, or they could be sharp but fixed, reading the same mixture every time, in which case the mechanism is a learned pooling rather than a conditional one. This section tests both cases.

C.2.1. Attention Concentration

Write β for one account view's weight vector over the n elements it attends to, either the six feature keys or the transactions in its window. We measure concentration using the top- k mass, i.e., the sum

of the k largest entries of β , and the effective number of keys:

$$H = - \sum_{i=1}^n \beta_i \ln \beta_i, \quad N_{\text{eff}} = \exp(H),$$

where H is the entropy of the model’s actual attention weights. N_{eff} is the number of equally weighted keys that would give the same spread. Thus, uniform attention gives $N_{\text{eff}} = n$, while putting all weight on one element gives $N_{\text{eff}} = 1$. We measure differences between weight vectors using total variation (TV):

$$\text{TV}(q, r) = \frac{1}{2} \sum_{i=1}^n |q_i - r_i|.$$

For the feature keys, Table C.1 reports $\text{TV}(\beta, \bar{\beta})$ averaged over every row, each row’s own weight vector against the class mean $\bar{\beta}$. TV is not reported for the windows because different windows contain different numbers of transactions and therefore do not share a common vector space.

Table C.1: Concentration of both attention mechanisms in the full TCEE-MLP. *Elem.* is the number of available keys. N_{eff} is the effective number of keys. TV is the distance from the class mean and is reported only for feature keys because windows have different lengths.

Attention	Class	Rows	Elem.	N_{eff}	Top-1	Top-3	TV
Feature keys	Illicit	3 222	6	4.63 ± 0.18	0.374 ± 0.019	0.774 ± 0.020	0.237 ± 0.044
	Licit	10 000	6	4.78 ± 0.23	0.349 ± 0.019	0.761 ± 0.032	0.173 ± 0.030
Window, whole	Illicit	3 222	23.2	13.17 ± 3.47	0.312 ± 0.094	0.550 ± 0.112	—
	Licit	10 000	27.3	18.78 ± 3.22	0.226 ± 0.053	0.412 ± 0.080	—
Window, context only	Illicit	3 207	22.3	13.21 ± 3.02	0.306 ± 0.085	0.546 ± 0.100	—
	Licit	9 996	26.3	18.38 ± 2.82	0.238 ± 0.050	0.412 ± 0.071	—

Table C.1 shows that neither attention mechanism is uniform, but neither is highly concentrated either. For the six feature keys, N_{eff} is about 4.6–4.8, and the largest key receives about 35–37% of the weight. The top three keys receive about 76–77%.

Window attention is more concentrated for illicit transactions in raw terms: $N_{\text{eff}} = 13.21$ for illicit transactions versus 18.38 for licit transactions. However, licit windows are also longer on average (26.3 versus 22.3 transactions). After normalising N_{eff} by window length, the effective coverage is 0.612 ± 0.123 for illicit transactions and 0.710 ± 0.104 for licit transactions. The difference is smaller than the variation across random seeds, so the difference in window concentration between illicit and licit classes is not clear. The feature-key difference is even smaller (4.63 versus 4.78).

For feature keys, TV is 0.237 for illicit transactions and 0.173 for licit transactions. Thus, the model does not use exactly the same feature-key mixture for every transaction, but the variation is small.

Finally, we include the scored transaction in the window (window whole row). Its attention weight is 0.114 for illicit transactions and 0.042 for licit transactions. Removing it changes the window top-1 attention only slightly, from 0.312 to 0.306 for illicit transactions. This suggests that the concentration pattern is driven by the surrounding context rather than by the scored transaction itself.

C.2.2. Account-Level and Transaction-Level Variation

To determine whether the variation between transaction weights is mainly caused by account differences or by transaction-specific behaviour, we consider accounts with at least two distinct transactions. Let N be the number of retained transactions, $\beta^{(t)}$ the feature-key weight vector for transaction t , $\bar{\beta}^{(a)}$ the mean vector for account a , and $a(t)$ the account associated with transaction t . We split the variation into between-account and within-account components:

$$\text{TV}_{\text{between}} = \frac{1}{N} \sum_{t=1}^N \text{TV}(\bar{\beta}^{(a(t))}, \bar{\beta}), \quad \text{TV}_{\text{within}} = \frac{1}{N} \sum_{t=1}^N \text{TV}(\beta^{(t)}, \bar{\beta}^{(a(t))}).$$

We also report the exact fraction of variance explained by account identity:

$$\text{var}_{\text{between}} = 1 - \frac{\sum_{t=1}^N \|\beta^{(t)} - \bar{\beta}^{(a(t))}\|^2}{\sum_{t=1}^N \|\beta^{(t)} - \bar{\beta}\|^2}.$$

Here, $\bar{\beta}$ is the class mean for the two class rows and the pooled mean for the *All* row. Thus, the *All* row also captures differences between the two class means.

Table C.2: Between-account and within-account variation in feature-key attention for accounts with at least two distinct transactions. *Views* is the number of retained account views, *Accts* is the number of accounts, and $\text{var}_{\text{between}}$ is the fraction of variance explained by account identity.

Class	Views	Accts	Between	Within	$\text{var}_{\text{between}}$
Illicit	1 667	385	0.209 $\pm$ 0.042	0.044 $\pm$ 0.010	0.923 $\pm$ 0.049
Licit	889	225	0.103 $\pm$ 0.010	0.017 $\pm$ 0.004	0.942 $\pm$ 0.030
All	2 660	639	0.186 $\pm$ 0.037	0.035 $\pm$ 0.007	0.930 $\pm$ 0.046

Table C.2 shows that most of the variation in feature-key attention is between accounts rather than between transactions within the same account. The between-account TV is 0.209 for illicit transactions and 0.103 for licit transactions, while the within-account TV is only 0.044 and 0.017, respectively. Account identity explains more than 92% of the variance in all three rows.

Thus, transactions from the same account tend to receive very similar feature-key attention, even when their classes differ. The pooled *All* row has a within-account TV of only 0.035. The feature retrieval is therefore primarily account-level rather than strongly transaction-specific.

C.2.3. Feature-wise Attention

This part reads the plain TCEE-MLP, the same checkpoint the next section retrains and intervenes on, so the weights and the importance ordering it is later compared against come from one model. Each transaction is scored twice, once as a retrieval over the sender’s own history and once over the receiver’s, so the two views are kept separate rather than pooled. Figure C.1 shows the mean weight each view gives to each of the six keys, against 0.167, the weight an indifferent retrieval would give every key.

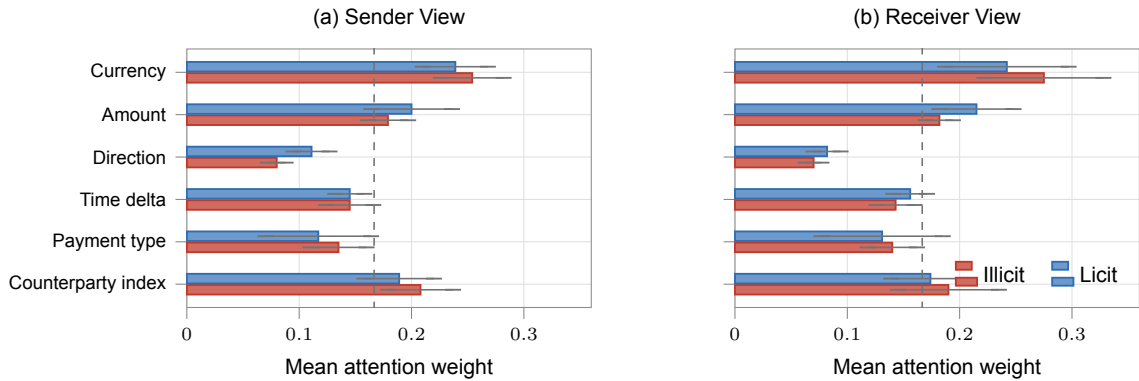


Figure C.1: Feature attention per input series, plain TCEE-MLP. The dashed line at 0.167 is the weight an indifferent retrieval would give each of the six keys. The two panels are the sender and receiver views of the same transactions.

Ranking the six keys separately for each seed, view and class gives $5 \times 2 \times 2 = 20$ rankings in total. Only two keys are stable across ranks. Currency is one of the two heaviest in 19 of the 20, and direction is one of the two lightest in all 20. The other four, counterparty index, amount, time delta and payment type, switch places from seed to seed. The differences between licit and illicit are mostly within seed noise, so no reliable conclusion can be drawn about which features draw more attention weight for one class than the other.

C.3. Attention and Feature Importance

Attention selecting a feature does not mean the model needs it. This section first establishes what the model actually needs by retraining without each feature, then tests the attention weights against what that retraining found by intervening on the model directly.

C.3.1. What the Model Needs

We remove each of the six input series in turn and retrain the plain TCEE-MLP, so the series is absent from the Chronos input and from the transaction query alike. The reported spread is of the paired, seed-by-seed difference.

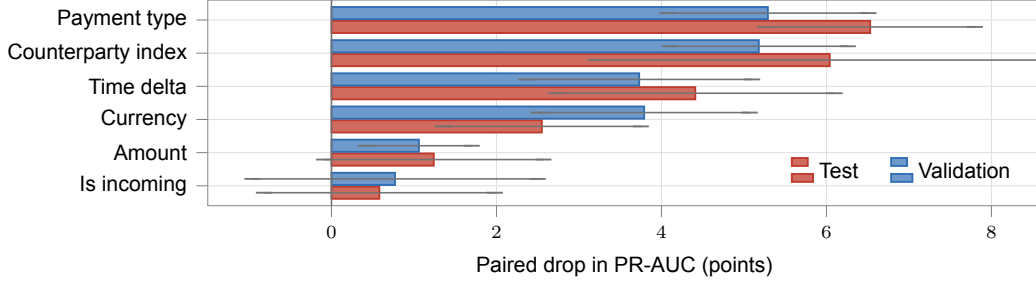


Figure C.2: Leave one covariate out on HI-Small, plain TCEE-MLP. Each bar is the paired drop from the full-feature control, which scores 69.24 ± 0.61 on test and 56.53 ± 0.98 on validation. The whisker is the spread of the paired difference across seeds, so a whisker reaching past zero means the drop is inside seed noise.

Figure C.2 puts payment type and the counterparty index first at 6.53 and 6.04 points, followed by the time delta at 4.41 and currency at 2.55. Removing the amount produces a smaller and less consistent change, and the incoming flag has no measurable effect. Test and validation disagree only on whether currency sits above the time delta. The test ordering is the one the next two sections test the attention against.

C.3.2. Feature Attention Deletion

We ask whether removing what the attention points to changes the prediction more than removing other features. There are two ways a feature can influence the prediction, through Chronos and through the transaction query. Instead of retraining the model $2^6 = 64$ times with every possible subset of features, we can mask out a feature from the attention softmax and replace its value in the transaction query with the pool average for that feature. Notably, because of Chronos' group attention mechanism the model can still encode information about a feature in another feature's representation. This means that masking only blocks the attention from reading that feature explicitly and the results should be read accordingly.

We add two controls to make the test meaningful, deleting the same number of features at random and deleting the lowest weighted ones, because removing any feature should hurt the prediction somewhat, and the question is whether removing what the model attends to hurts more than that.

Write m_{full} for the illicit logit minus the licit logit, with nothing deleted, and call this difference the margin. m_k is the same margin once a set of k features has been deleted. Damage is

$$\delta_k = \begin{cases} m_{\text{full}} - m_k & y = 1 \\ m_k - m_{\text{full}} & y = 0 \end{cases} \quad (\text{C.1})$$

interpreted as the movement of the margin towards the wrong class, signed by the label y so that a positive number means the deletion hurt the prediction in both classes, and a negative one means the deletion moved the margin towards the correct class instead, so the prediction became more confident once the feature was gone.

Figure C.3 shows that the feature attention does not consistently identify the features whose removal most damages the prediction. On illicit transactions, deleting the single highest weighted feature does 0.70 of damage while deleting a random one does 1.45, and the attention ordering stays at or below

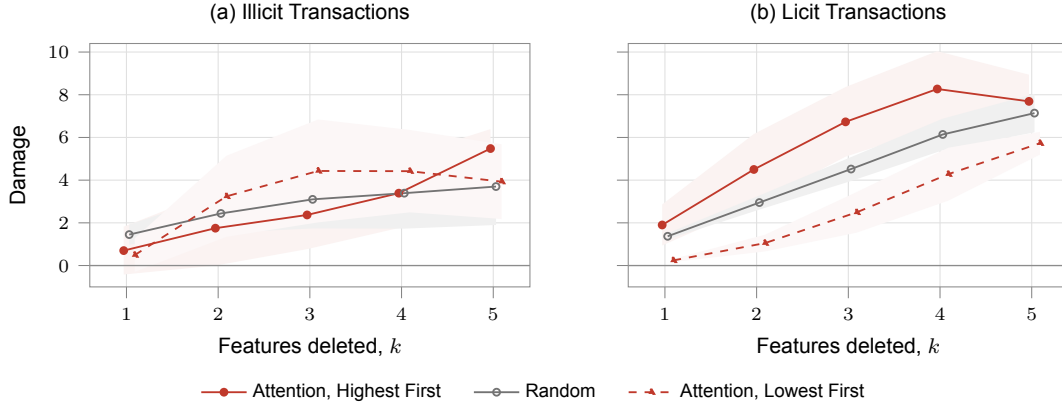


Figure C.3: Deleting features in three orders, plain TCEE-MLP. Damage is the movement of the margin (the illicit logit minus the licit logit) towards the wrong class after deleting each transaction's own top- k .

the random control through $k = 4$. On licit transactions it runs the other way and beats random at every k , 1.90 against 1.37 at $k = 1$. For $k=6$ with every feature deleted, there is nothing left for the attention softmax to normalise over so it is not shown. Taken together, these results suggest that the feature weights track importance more clearly for licit transactions, while they do not provide a useful importance ordering for illicit transactions.

C.3.3. Neighbourhood Attention Deletion

The same test on the neighbourhood window means turning off a transaction's mask so that the model sees a shorter window. The scored transaction is protected in every condition, since it is the target and hiding it would bias the results. Every scored transaction in the test split is rescored with its k highest attention transactions deleted from both endpoint windows, with its k lowest deleted and with k random ones deleted, for k in 1, 2, 4, 8. The two kinds of curve draw on different populations. Damage needs the same transaction scored under every k and only a transaction whose weaker endpoint holds at least eight context transactions can receive the full deletion at every k , so the damage curves keep 88% of the licit transactions and 65% of the illicit ones. PR-AUC is measured on the full split, because for accounts with fewer than k transactions we can simply delete all of them.

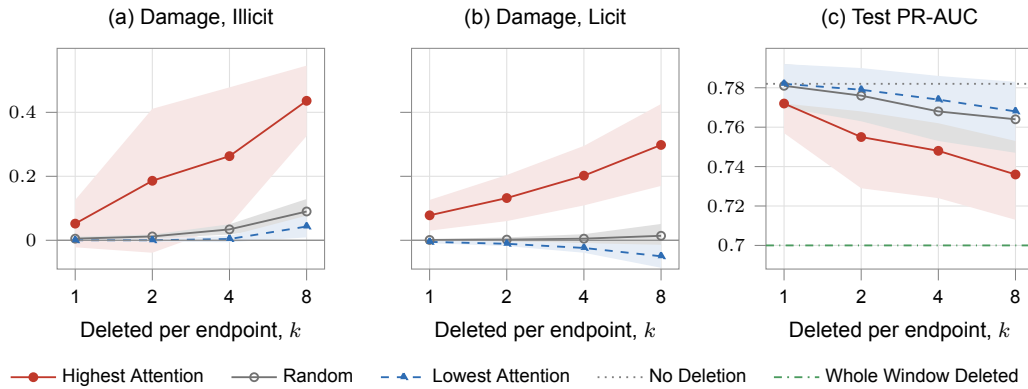


Figure C.4: Deleting context transactions from the neighbourhood window, full TCEE-MLP. Damage is the movement of the margin (the illicit logit minus the licit logit) towards the wrong class, signed by the label. The scored transaction is kept in every condition.

The window weights behave differently from the feature weights in this deletion test, as Figure C.4 shows. Deleting the highest attended context hurts more than deleting random context at every k , and the gap widens as k grows. Deleting the lowest attended context does essentially no damage, faintly negative on licit transactions and near zero on illicit ones.

C.4. Attention and Pattern Identification

The attention mechanism being able to select transactions that matter for higher prediction is not necessarily the same as being able to select transactions that matter for an analyst. In this section, we test whether the model’s attention weights can help an analyst find the other transactions in the same pattern of the scored transaction.

C.4.1. Identifying Pattern-Related Transactions

AMLWorld labels the pattern an illicit transaction belongs to. In this section, we test whether the model’s attention weights can help an analyst find the other transactions in the same pattern of the scored transaction, and whether the attention weights are higher because of proximity to the scored transaction or because of pattern membership. To test this, we bin transactions by their time difference $|t_i - t_e|$ into seven roughly logarithmic bins with edges at one minute, ten minutes, one hour, six hours, one day and one week, and the weights are reshuffled within each bin 200 times. The bins ignore before-versus-after position. In Table C.3, *Mass* is the share of attention weight on sibling transactions, and *Null* is the same quantity after the shuffle. The difference between the two is the lift, which is the share of attention weight on sibling transactions that cannot be explained by proximity alone.

Table C.3: Attention on the other transactions in the scored transaction’s own pattern, full TCEE-MLP. *Win.* is the number of account-endpoint windows analysed. *Cov.* is the share of them holding at least one sibling transaction, and the remaining columns average over those. *Txns* is the mean number of sibling transactions in a window. *Share* is the mean fraction of a window’s transactions that are siblings.

Typology	Win.	Cov.	Txns	Mass	Share	Null	Lift
Gather-scatter	685	0.53	12.2	0.538 $\pm$ 0.102	0.414	0.440 $\pm$ 0.025	0.099 $\pm$ 0.078
Fan-in	239	0.49	9.8	0.764 $\pm$ 0.040	0.476	0.619 $\pm$ 0.020	0.145 $\pm$ 0.025
Fan-out	231	0.50	8.4	0.389 $\pm$ 0.058	0.285	0.302 $\pm$ 0.015	0.086 $\pm$ 0.048
Scatter-gather	410	0.99	5.1	0.345 $\pm$ 0.048	0.235	0.275 $\pm$ 0.027	0.070 $\pm$ 0.023
Stack	189	0.50	1.0	0.102 $\pm$ 0.030	0.083	0.070 $\pm$ 0.017	0.031 $\pm$ 0.013
Random	148	0.81	1.0	0.120 $\pm$ 0.025	0.073	0.100 $\pm$ 0.021	0.021 $\pm$ 0.007
Cycle	172	0.99	1.0	0.107 $\pm$ 0.020	0.066	0.079 $\pm$ 0.016	0.029 $\pm$ 0.006
Bipartite	120	0.00	–	–	–	–	–
All	2 194	0.63	6.5	0.369 $\pm$ 0.048	0.261	0.296 $\pm$ 0.019	0.073 $\pm$ 0.029

Sibling transactions receive 36.9% of the attention, while the shuffle gives them 29.6%. This suggests that some of the attention on sibling transactions is related to pattern membership, rather than just their being close in time to the scored transaction. The lift is around +0.07 to +0.15 for fan-in, fan-out, gather-scatter and scatter-gather, where an account contains several transactions from the same pattern. For stack, random and cycle, where it contains only one, the lift is smaller, at around +0.02 to +0.03. The variation across seeds is large enough that we cannot draw conclusions from differences between specific typologies.

Overall, the results suggest a small effect beyond temporal proximity, mainly when an account contains several transactions from the same pattern. When there is only one sibling transaction, the effect is much weaker. These rows are also harder to interpret because the shuffle can only move a sibling when other transactions fall in the same $|t_i - t_e|$ bin. More generally, these results are limited to transactions that have an assigned pattern, and it is not clear how they would generalise to unknown patterns.