A Comparative Study of Temporal Graph Networks and Sequential Profilers for Money Laundering Detection on Imbalanced Data

1st Anatol Satler
School of Electronic Engineering and Computer Science
Queen Mary University
London, United Kingdom
a.satler@se24.qmul.ac.uk

***Abstract*—Detecting money laundering is a critical challenge for financial institutions, complicated by extreme class imbalance and the sophisticated, evolving nature of illicit activities. Deep learning offers promising solutions, but the optimal architectural approach remains an open question. This paper presents a rigorous comparative study of two distinct deep learning paradigms for this task: a graph-based approach using a Dynamic Temporal Graph Convolutional Network (GCN) designed to model localized transactional structures, and a sequence-based approach using an End-to-End Sequential Profiler with a Bidirectional GRU, designed to learn the behavioral signatures of individual accounts over time. We conduct extensive experiments on a realistic, large-scale synthetic dataset. Our results reveal a evident performance difference: the sequential profiler achieves an Area Under the Precision-Recall Curve (AUPRC) of 0.1622, a nearly seven-fold improvement over the GCN's 0.0235. This decisive outcome suggests that for detecting the laundering patterns in this dataset, learning long-range temporal dependencies within an account's history is a significantly more powerful signal than modeling its immediate, local graph-based interactions.**

***Index Terms*—Anti-Money Laundering (AML), Deep Learning, Graph Neural Networks, Recurrent Neural Networks, Imbalanced Data, Anomaly Detection**

I. Introduction

Money laundering poses a significant threat to global financial stability and security. The United Nations Office on Drugs and Crime (UNODC) estimates that the amount of money laundered globally in one year is between 2% and 5% of global GDP, which corresponds to $800 billion to $2 trillion (United Nations Office on Drugs and Crime (UNODC) n.d.). Financial institutions are mandated to deploy Anti-Money Laundering (AML) systems, but traditional rule-based methods are often ineffective, generating high volumes of false positives and failing to adapt to novel criminal typologies. Consequently, there is a growing interest in applying advanced machine learning and deep learning techniques to improve detection accuracy and operational efficiency. However, the application of deep learning to AML is fraught with challenges, including a severe scarcity of public, labeled data, extreme class imbalance where illicit transactions are exceedingly rare, and the need for model interpretability to satisfy regulatory requirements (Kute et al. 2021). To model financial transactions, two dominant paradigms have emerged: graph-based methods, which represent accounts as nodes and transactions as edges to capture relational structures (Weber et al. 2019), and sequence-based methods, which model the chronological history of account activity to learn behavioral patterns (Yu et al. 2024). While both approaches have theoretical merit, their relative effectiveness, particularly on highly imbalanced and dynamic datasets, has not been thoroughly compared. This study addresses this gap through a direct comparison of these architectural philosophies. Our contributions include: 1) the design and implementation of a Dynamic Temporal GCN (Model A) for relational patterns and an End-to-End Sequential Profiler (Model B) for behavioral signatures; 2) a comprehensive evaluation on the large-scale synthetic AMLworld dataset against multiple baselines; and 3) a demonstration that the sequential model dramatically outperforms not only the graph-based model but also a strong, feature-engineered baseline from the dataset's creators. We conclude by analyzing the practical implications of these findings, arguing that temporal signals are more potent than localized structural ones for this task.

II. Background

This section provides essential context for our study. It begins by reviewing the state of deep learning applications in Anti-Money Laundering (AML) and then establishes the foundational concepts of Graph Neural Networks (GNNs), which form the basis for one of the two architectures central to our comparative analysis.

A. Challenges and Approaches in Anti-Money Laundering

The application of Deep Learning (DL) to the complex domain of Anti-Money Laundering (AML) has seen growing interest, though it is fraught with challenges. A critical review by Kute et al. (2021) highlights the central obstacles hindering progress: a significant lack of model interpretability required for regulatory compliance, a scarcity of labeled real-world data, and extreme class imbalance in transaction datasets. The review notes that while methods like Convolutional Neural Networks (CNNs) and Autoencoders are being explored,

proprietary industry solutions remain a black box, and the immense legal and privacy barriers often prevent the academic-industry collaborations needed to advance the field.

To address the critical lack of realistic public data, Altman et al. (2023) introduced "AMLworld," an advanced, agent-based synthetic transaction generator. This framework models the entire money laundering cycle and produces datasets with complete ground-truth labels, a key advantage over incomplete real-world data. They provide baseline results using GNNs, demonstrating that sharing data across simulated banks can significantly improve detection accuracy. However, the utility of this dataset is bounded by its synthetic nature. The pre-programmed laundering patterns may not fully capture the adversarial and dynamic behavior of real criminals, potentially making trained models brittle. Furthermore, the focus on a multi-bank collaboration scenario is highly unrealistic in practice due to competition and privacy regulations, and the proprietary nature of the generator itself limits the community's ability to create variations.

Building on the potential of graph-based methods, early work by Weber et al. (2019) established the use of Graph Convolutional Networks (GCNs) for financial forensics on the Bitcoin-based Elliptic dataset. As a foundational "first look," the work demonstrated a proof-of-concept but also revealed limitations; its simplified simulator, "AMLSim," only models the "layering" phase of money laundering, and the proposed model's scalability was insufficient for real-time, low-latency production systems. More recently, Yu et al. (2024) proposed a hybrid CNN-GRU model for anomaly detection on the same dataset, reporting high accuracy with a contrastive learning approach. Reproducing these results, however, is challenging due to significant methodological ambiguities. A central weakness is the lack of clarity in their contrastive learning sampling strategy. The authors state that positive pairs are selected based on having "similar" transaction frequency and amount, but they do not provide a quantifiable metric or a specific algorithm for this selection, making the process subjective and difficult to replicate.

B. Foundations of Graph Neural Networks

The core technology for modeling relational data like transaction networks is the Graph Neural Network (GNN). The foundational GNN model was introduced by Scarselli et al. Scarselli et al. (2009) as a framework for processing general graph structures. Its core concept is an information diffusion mechanism where each node's state is updated based on its neighbors until a stable equilibrium or fixed point is reached. The state of a node n, denoted x_n ($x_n \in \mathbb{R}^d$), is updated via a parametric function f_w:

$$x_n = f_w(l_n, l_{\text{co}[n]}, x_{\text{ne}[n]}, l_{\text{ne}[n]}) \tag{1}$$

where l_n and $l_{\text{ne}[n]}$ are the labels of the node and its neighbors, and $x_{\text{ne}[n]}$ are the states of its neighbors. l are constant vectors of information while $x_{\text{ne}[n]}$ represent the information that is being forwarded in each iteration of $l_{\text{ne}[n]}$. Convergence to a unique fixed point is guaranteed by constraining the global transition function to be a contraction map, in Euclidean distance. Despite its theoretical importance, this original formulation has significant limitations, including high computational cost due to the iterative relaxation process for every forward pass and scalability issues, which have led to it being largely superseded by more efficient architectures.

To overcome these scalability bottlenecks, Kipf and Welling Kipf & Welling (2016) introduced the Graph Convolutional Network (GCN), a highly scalable and effective architecture that has become a foundational method in graph machine learning. The GCN simplifies the information diffusion process into an efficient layer-wise propagation rule:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{2}$$

H is a node matrix each row represents a node, while every column represents a feature ($H \in \mathbb{R}^{N \times d_{feature}}$). The aggregation part is: $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$, where $\tilde{D}^{-\frac{1}{2}}$ is a diagonal degree matrix representing the square-rooted edge weight of the graph. The model aggregates feature information from a node's immediate neighbors in a single, non-iterative step per layer, using a symmetrically normalized adjacency matrix with self-connections ($\tilde{A} = A + I$). The GCN is motivated as a localized, first-order approximation of spectral graph convolutions. However, the GCN model itself is not without constraints. Its original formulation is transductive, a paradigm that limits its application on dynamic graphs with unseen nodes. In this setting, the model learns embeddings for a fixed set of nodes seen during training and cannot generate predictions for new entities introduced post-training. It is unable to create an embedding for a new account without re-running the entire training process on a graph that includes the new node. Our temporal snapshot approach, explained later, is designed to overcome this specific limitation by operating on small, dynamic subgraphs Kazemi et al. (2020). Furthermore, it is prone to oversmoothing in deep architectures and operates under an implicit assumption of homophily Ma et al. (2021). A graph exhibits homophily when:

$$\forall (i,j) \in E, h_i \approx h_j \text{ or } y_i = y_j \tag{3}$$

That is, for most $edges(i,j)$, either the node features h_i,h_j are similiar, or the labels y_i, y_j are the same, meaining neighbors are similiar. In our case, it is a mixture of homophily and heterophily because of the combinations of Licit-to-Licit connection and Illicit-to-Licit and so on.

III. Methodology

This chapter details the experimental methodology, we used in this research. We first describe the dataset and the data preprocessing steps common to both approaches. We then provide a detailed architectural breakdown of the two models developed for this comparative analysis: the **Dynamic Temporal GCN (Model A)** and the **End-to-End Sequential Profiler (Model B)**. Finally, we outline the training procedures and evaluation metrics used to assess and compare their performance.

A. Dataset

The experiments in this study were conducted using the `LI-Small_Trans.csv` dataset from the AMLworld framework (Altman et al. 2023), a synthetic financial transaction dataset designed to simulate realistic money laundering patterns. The dataset comprises 6,924,049 transactions involving a total of 705,903 unique accounts. Each transaction is described by 11 features, including timestamp, sender/receiver information, amount, and payment format. The dataset exhibits a severe class imbalance characteristic of real-world AML scenarios, with only 3,565 transactions labeled as illicit, accounting for just 0.0515% of the total volume. This extreme rarity of positive samples, corresponding to an imbalance ratio of approximately 1941:1, makes it a challenging and realistic benchmark for evaluating AML detection models.

B. Preprocessing

The raw tabular dataset is transformed into a set of chronologically ordered NumPy arrays. First, all unique sender and receiver account strings are mapped to unique integer IDs, a requirement for GNNs. Transaction features are then engineered: timestamps are converted to cyclical sine/cosine features to preserve the continuity of a day, transaction amounts are log-transformed to serve as edge weights and mitigate the influence of outliers, and categorical features like payment type are one-hot encoded. Crucially, the entire dataset is sorted by timestamp to enable the creation of meaningful temporal snapshots. Finally, all relevant columns are converted to NumPy arrays for efficient processing.

Mathematical Formulation: Let the raw dataset be a sequence of M transactions, $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_M\}$, where each transaction τ_i is a tuple containing raw attributes like sender, receiver, amount, timestamp, etc.

- **Feature and Node definition:** Let A the set of all unique accounts in the dataset. We define a bijective mapping $f_{id} : A \to \{0, 1, ..., N-1\}$, where $N = |A|$, for any transaction τ_i with sender s_i^{raw} and receiver r_i^{raw}, their integer IDs are $s_i = f_{id}(s_i^{raw})$ and $r_i = f_{id}(r_i^{raw})$.
  For each transaction τ_i, we construct a feature vector. The timestamp t_i is converted to its hour $h_i \in [0, 23]$. The temporal feature is:

$$x_{time,i} = \left[sin\left(\frac{2\pi h_i}{24}\right), cos\left(\frac{2\pi h_i}{24}\right) \right] \in \mathbb{R}^2 \quad (4)$$

  The transaction amount a_i is transformed into a scale edge weight:

$$w_i = log(1 + a_i) \in \mathbb{R} \quad (5)$$

  Categorical feature are one-hot encoded into a binary vector $x_{cat,i} \in \{0, 1\}^{d_{cat}}$. These feature are concatenated to form the transaction's edge attribute vector, $\mathbf{x}_{attr,i} \in \mathbb{R}^{d_{attr}}$ (e.g. $x_{cat} = [0, 1, 0, 0, 1]$, where each column represent another attribute).

$$\mathbf{x}_{attr,i} = \mathbf{x}_{time,i} \oplus \mathbf{x}_{cat,i} \oplus ... \quad (6)$$

- **Sorting and Final Output** The sequence $\mathcal{T}$ is sorted by timestamp, yielding an ordered sequence $\mathcal{T}'$. The preprocessing function outputs the following NumPy arrays, which are sequences derived from $\mathcal{T}'$:
  - Senders $S = (s_1, s_2, ..., s_m) \in \mathbb{Z}^M$
  - Receivers $R = (r_1, r_2, ..., r_m) \in \mathbb{Z}^M$
  - Labels $\mathbf{Y} = (y_1, y_2, ..., y_m) \in \{0, 1\}^M$
  - Edge Weights $\mathbf{M} = (w_1, w_2, ..., w_m) \in \mathbb{R}^M$
  - Edge Attributes $\mathbf{X}_{attr} \in \mathbb{R}^{M \mathbf{x} d_{attr}}$, where the i-th row is $X_{attr,i}^T$.

C. Model Architectures

We implemented two distinct model architectures, each embodying a different philosophical approach to graph-based anomaly detection.

*1) **Model A: Dynamic Temporal GCN**:* Our first approach, Model A, is a **Dynamic Temporal Graph Convolutional Network (Temporal GCN)**, designed to learn from localised, time-sensitive behavioural patterns. Its core principle is to operate not on a single static graph, but on a series of dynamic **temporal graph snapshots**. Each snapshot, denoted $\mathcal{G}_j$, is generated for a target transaction τ_j and is comprised of τ_j itself plus the k transactions that occurred immediately prior. The complete end-to-end architecture is visualized in Fig. 1. As the diagram illustrates, the model processes each snapshot through three distinct stages: (1) generating initial vector representations for the accounts and the target transaction; (2) enriching these representations through a stack of GCN layers to capture relational context; and (3) assembling a final predictive vector for a downstream MLP classifier. The following sections detail each of these stages.

The model architecture processes each snapshot $\mathcal{G}_j$ through the following stages:

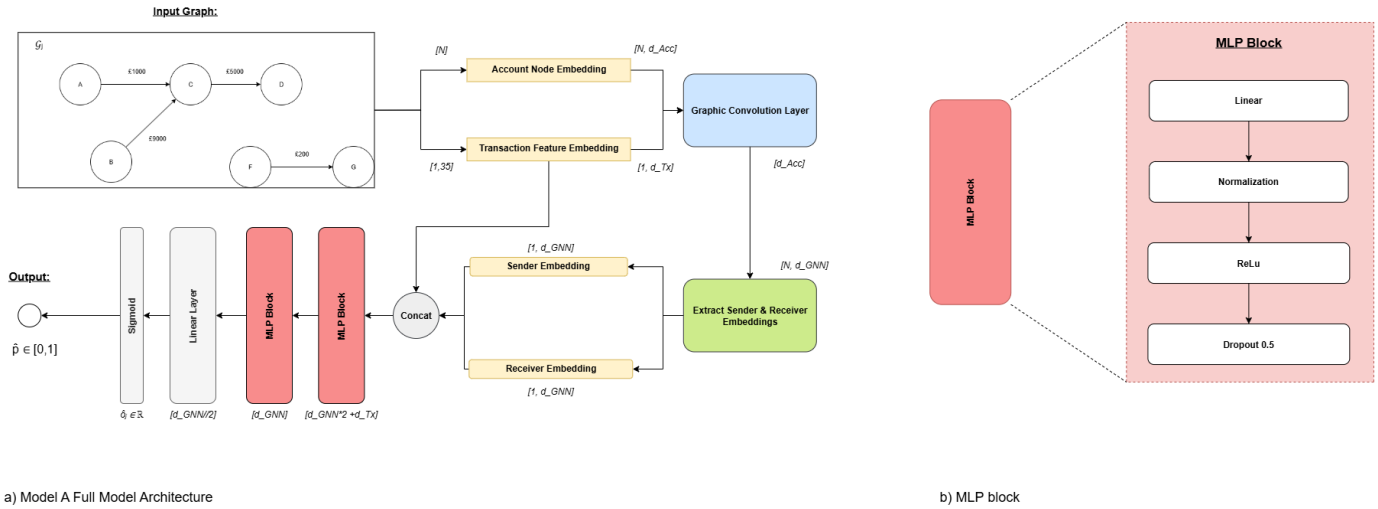
*a) **1. Initial Feature Representation (Embedding)**:* The model first generates initial vector representations for the nodes and the target transaction within the snapshot.

- **Node Embeddings:** A global, learnable account embedding matrix $\mathbf{E}_{\mathcal{A}} \in \mathbb{R}^{N_{\text{accounts}} \times d_{\text{acc}}}$ is maintained, where N_{accounts} is the total number of unique accounts in the dataset and d_{acc} is the embedding dimension. For a given snapshot $\mathcal{G}_j$, the initial node feature matrix $\mathbf{H}_j^{(0)}$ is formed by retrieving the corresponding vectors from $\mathbf{E}_{\mathcal{A}}$ for every node in $\mathcal{V}_j$.

$$\mathbf{H}_j^{(0)} = \text{Lookup}(\mathbf{E}_{\mathcal{A}}, \text{IDs}(\mathcal{V}_j)) \in \mathbb{R}^{d_{\text{acc}}} \quad (7)$$

  Lookup is a function that retrieves information from a node with its features (a vector).
- **Transaction Embedding:** The feature vector of the target transaction, $\mathbf{x}_{\text{attr},j}$, is projected into a dense embedding space using a dedicated linear layer with trainable weights $\mathbf{W}_{\text{tx}}$ and bias $\mathbf{b}_{\text{tx}}$.

$$\mathbf{e}_j = \text{ReLU}(\mathbf{W}_{\text{tx}} \mathbf{x}_{\text{attr},j} + \mathbf{b}_{\text{tx}}) \in \mathbb{R}^{d_{\text{tx}}} \quad (8)$$



a) Model A Full Model Architecture

b) MLP block

Fig. 1. Architecture of Model A (Dynamic Temporal GCN). (a) A graph snapshot G_j is processed via embedding and graph convolution layers. Context-aware sender and receiver embeddings are concatenated with the transaction embedding and passed to an MLP classifier to produce a prediction $\hat{p}$. (b) The internal structure of the reusable MLP Block.

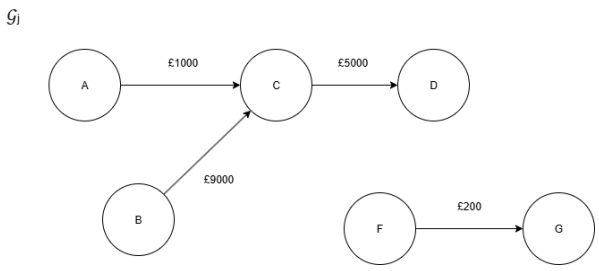


Fig. 2. $\mathcal{G}_j$ example

*b) **2. Graph Convolutional Propagation**:* The core of the model is a stack of L Graph Convolutional Network (GCN) layers. These layers iteratively update the initial node embeddings, enriching them with structural information from their local neighborhood within the snapshot. The propagation rule for layer l is defined as:

$$\mathbf{H}_j^{(l+1)} = \text{Dropout}\left(\text{ReLU}\left(\hat{\mathbf{D}}_{w,j}^{-\frac{1}{2}}\hat{\mathbf{A}}_{w,j}\hat{\mathbf{D}}_{w,j}^{-\frac{1}{2}}\mathbf{H}_j^{(l)}\mathbf{W}^{(l)}\right)\right) \tag{9}$$

where $\mathbf{H}_j^{(l)}$ is the matrix of node representations at layer l, and $\mathbf{W}^{(l)}$ is the layer's trainable weight matrix. The matrix $\hat{\mathbf{A}}_{w,j} = \mathbf{A}_{w,j} + \mathbf{I}$ is the weighted adjacency matrix of the snapshot with added self-loops. The identity matrix secures the information of its own node, and $\hat{\mathbf{D}}_{w,j}$ is its diagonal degree matrix. This formulation allows the model to modulate the influence of neighboring nodes based on the financial magnitude of their interaction, as encoded in the edge weights. After L layers, the resulting matrix $\mathbf{H}_j^{(L)}$ contains context-aware embeddings for every node in the snapshot.

In Fig.2, we can see a small example of how $\mathcal{G}_j$ could look like. In this example, if the transaction C→D were the target for classification, the model's task is to determine its legitimacy by analysing its context within this snapshot. A single GCN layer ($L = 1$) would allow node D to aggregate information solely from its direct predecessor, node C. At this stage, node D's representation would be context-aware but blind to the origin of the funds received by C. Becoming h_{c_1} and h_{d_1} simultaneously, these represent their learned first-layer versions.

A second propagation step ($L = 2$), however, fundamentally expands this receptive field. During this second pass, node D again aggregates from node C, but it now receives C's updated representation from the first layer (h_{c_1}), which is already impregnated with information from nodes A and B. As a result, the final embedding for node D becomes sensitive to the full 2-hop path, effectively allowing the model to "see" the complete layering patterns A→C→D and B→C→D, becoming h_{d_2}.

Furthermore, the influence of these paths is modulated by the edge weights; the information path originating from node B (with an incoming transaction of £9000) would have a stronger impact on C's, and subsequently D's, representation than the path from node A (£1000). This mechanism enables the model to learn representations that distinguish between simple, isolated transactions (such as F→G) and more complex, potentially suspicious topological motifs, thereby forming the basis for its classification decision.

*c) **3. Predictive Vector Assembly and Classification**:* Following the graph propagation stage, the model constructs a final, comprehensive feature vector specifically for the target transaction τ_j. This is achieved by first extracting the final, contextualized embeddings for the transaction's sender (s_j) and receiver (r_j) from the updated node representation matrix:

- **Sender Embedding** $\mathbf{h}_{s_j}^{(L)}$: The row in $\mathbf{H}_j^{(L)}$ corresponding to sender s_j.
- **Receiver Embedding** $\mathbf{h}_{r_j}^{(L)}$: The row in $\mathbf{H}_j^{(L)}$ corresponding to receiver r_j.

These two context-aware node embeddings are then concatenated with the target transaction's own embedding, $\mathbf{e}_j$, to form

the final predictive vector $\mathbf{z}_j$:

$$\mathbf{z}_j = \mathbf{h}_{s_j}^{(L)} \oplus \mathbf{h}_{r_j}^{(L)} \oplus \mathbf{e}_j \in \mathbb{R}^{2d_{\text{GNN}}+d_{\text{tx}}} \tag{10}$$

where $\oplus$ denotes concatenation. This vector represents the full context of the transaction, including the sender's role, the receiver's role, and the transaction's intrinsic properties.

Finally, $\mathbf{z}_j$ is passed to a classification head, which is a Multi-Layer Perceptron (MLP) composed of linear layers, batch normalization, and dropout. The MLP maps the feature vector to a single output logit, $\hat{o}_j$, representing the unnormalized prediction score for the transaction.

$$\hat{o}_j = f_{\text{MLP}}(\mathbf{z}_j; \theta_{\text{MLP}}) \in \mathbb{R} \tag{11}$$

This end-to-end architecture is then optimized using a standard loss function, such as weighted binary cross-entropy, to learn all model parameters ($\mathbf{E}_{\mathcal{A}}$, $\mathbf{W}_{\text{tx}}$, $\mathbf{W}^{(l)}$, and θ_{MLP}) simultaneously.

*2) **Model B: End-to-End Sequential Profiler**:* In stark contrast to the relational approach of Model A, our second model adapts the hybrid architecture proposed by Yu et al. (2024). It is an end-to-end sequential profiler designed to operate exclusively on the temporal dimension of transaction data. While the original work suggests a CNN-GRU combination, our implementation refines the initial feature embedding stage with a more direct Multi-Layer Perceptron (MLP), finding it better suited for the tabular feature structure. The model deliberately ignores the explicit graph structure and instead focuses on learning **behavioral signatures** from the ordered sequence of an account's activities. The core hypothesis is that illicit financial behavior manifests as anomalous temporal patterns that can be captured by a powerful sequence model.

*a) **1. Data Structuring for Sequential Analysis**:* While this model shares the same initial feature engineering as Model A (cyclical time features, one-hot encoding, etc.), the data is structured fundamentally differently. Instead of forming graph snapshots, we process the chronologically sorted transaction history of each account into a series of overlapping sequences.

For an account with a history of T transactions, $\{\tau_1, \tau_2, ..., \tau_T\}$, we generate a set of sequences using a sliding window of length L (sequence length) and a step size S. Each generated sequence $\mathcal{S}_i$ is a tuple containing both the feature vectors and their corresponding ground-truth labels:

$$\mathcal{S}_i = ((\mathbf{x}_i, \mathbf{x}_{i+1}, ..., \mathbf{x}_{i+L-1}), (y_i, y_{i+1}, ..., y_{i+L-1})) \tag{12}$$

where $\mathbf{x}_t \in \mathbb{R}^{d_{\text{feat}}}$ is the feature vector for transaction τ_t. This **sequence labeling** paradigm, where each transaction within a sequence retains its own label, is crucial. It allows the model to make a precise, per-transaction prediction while still leveraging the context of its temporal neighbors, avoiding the label dilution inherent in classifying the entire sequence with a single label.

*b) **2. Architecture: The MLP-GRU Behavioral Encoder**:* The heart of this model is a deep sequence encoder, depicted in Fig. 5, which processes each sequence of transaction features to produce a sequence of context-aware hidden states. It is composed of two primary stages:

- **Transactional Feature Embedding:** Each raw feature vector $\mathbf{x}_t$ in a sequence is first projected into a dense embedding space using a transaction-wise MLP. This initial step creates a richer, learned representation for each individual transaction before considering its sequential context.

$$\mathbf{e}_t = \text{MLP}(\mathbf{x}_t) \in \mathbb{R}^{d_{\text{tx}}} \tag{13}$$

- **Contextual Encoding with Bidirectional GRU:** The sequence of embedded transactions $(\mathbf{e}_1, ..., \mathbf{e}_L)$ is then processed by a multi-layer **bidirectional Gated Recurrent Unit (GRU)** Cho et al. (2014). We selected this architecture because its bidirectional nature is critical for the AML task. A suspicious transaction is often defined not only by its preparatory phase but also by subsequent actions, such as rapid layering across other accounts or immediate cash-outs, that are designed to obscure the funds' origin.

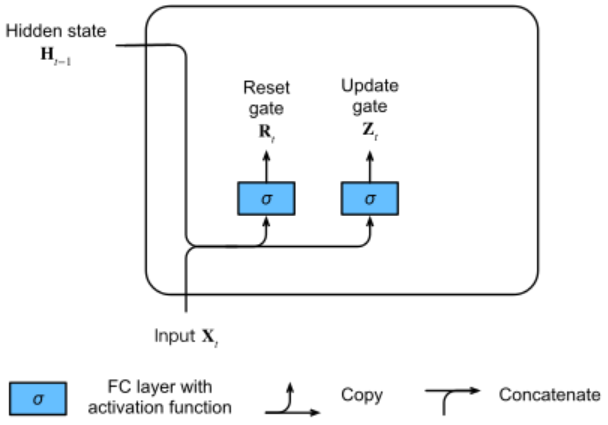


Fig. 3. The internal structure of a GRU cell, showing the computation of the reset and update gates. Adapted from Zhang et al. (2024).

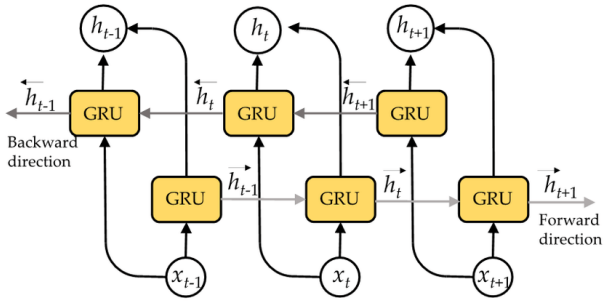
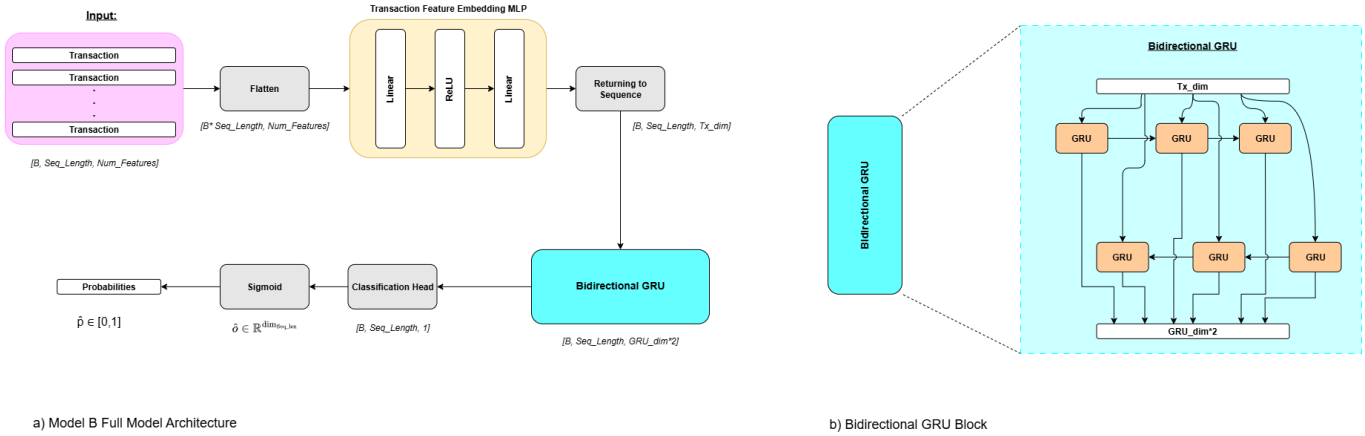


Fig. 4. Architecture of a Bidirectional GRU network, where an input sequence is processed in both forward and backward directions to capture past and future context. Adapted from Mekruksavanich & Jitpattanakul (2022).

A standard, unidirectional model would be blind to this crucial future context. This dual-pass architecture, which processes the sequence in both chronological and reverse-chronological order, is visualized in Fig. 4. The internal structure of the GRU cell (Fig. 3), with its learnable reset and update gates, enables the model to effectively



a) Model B Full Model Architecture

b) Bidirectional GRU Block

Fig. 5. Architecture of Model B (End-to-End Sequential Profiler). (a) A sequence of input transactions is first embedded individually by a transaction-wise MLP. The resulting sequence of embeddings is then processed by a Bidirectional GRU to capture temporal context. A final classification head produces a per-transaction probability score, $\hat{p}$. (b) The structure of the Bidirectional GRU, showing the parallel forward and backward passes.

learn these long-range dependencies while selectively discarding irrelevant information. By combining information from both directions, the model develops a much richer, holistic understanding of each transaction's context. The GRU outputs a hidden state for each timestep, which is the concatenation of its forward and backward states:

$$\overrightarrow{\mathbf{h}_t} = \text{GRU}_{\text{fwd}}(\mathbf{e}_t, \overrightarrow{\mathbf{h}_{t-1}}) \tag{14}$$

$$\overleftarrow{\mathbf{h}_t} = \text{GRU}_{\text{bwd}}(\mathbf{e}_t, \overleftarrow{\mathbf{h}_{t+1}}) \tag{15}$$

$$\mathbf{h}_t = \overrightarrow{\mathbf{h}_t} \oplus \overleftarrow{\mathbf{h}_t} \in \mathbb{R}^{2 \times d_{\text{gru}}} \tag{16}$$

The output of this stage is a sequence of context-aware embeddings, $\mathbf{H}_{\mathcal{S}} = (\mathbf{h}_1, ..., \mathbf{h}_L)$, where each vector $\mathbf{h}_t$ represents transaction τ_t enriched with information about its temporal neighborhood.

*c) **3. Per-Transaction Classification**:* The model's objective is classification. Therefore, each context-aware hidden state $\mathbf{h}_t$ from the GRU output is passed through a final linear layer (the classification head) to produce a single logit, $\hat{o}_t$, for each transaction in the sequence.

$$\hat{o}_t = \mathbf{w}_{\text{out}}^T \mathbf{h}_t + b_{\text{out}} \in \mathbb{R} \tag{17}$$

This architecture produces a sequence of predictions, $(\hat{o}_1, ..., \hat{o}_L)$, which are then compared against the ground-truth labels $(y_1, ..., y_L)$ using a weighted binary cross-entropy loss function. The entire encoder and classification head are **trained end-to-end**, optimizing all parameters simultaneously to minimize this loss. The final output provides a precise, per-transaction risk score informed by its sequential context.

D. Training and Evaluation

To ensure a fair and rigorous comparison, both the Dynamic Temporal GCN (Model A) and the End-to-End Sequential Profiler (Model B) were developed and assessed under a unified experimental framework.

*a) **MLP Baseline Model**:* To establish a clear performance baseline, we also implemented a simple Multi-Layer Perceptron (MLP) model. This model serves to quantify the performance gain achieved by incorporating contextual information (either relational or sequential). The MLP operates on the raw, flattened feature vector of each transaction in isolation, without any knowledge of its temporal or graph-based context. It consists of a simple architecture with hidden layers and is trained under the same conditions as our primary models, using the identical weighted binary cross-entropy loss function to ensure a fair comparison

*b) **Training Procedure**:* Both models were trained end-to-end using the Adam optimizer (Kingma & Ba 2014). To address the severe class imbalance inherent in financial transaction data, we employed a weighted binary cross-entropy (BCE) loss function. This loss function is critical for this task, as it increases the penalty for misclassifying the minority (illicit) class by assigning a higher weight to positive samples, calculated as the ratio of negative to positive instances in the training data. The dataset was chronologically split into training, validation, and test sets. All hyperparameter tuning was performed by optimizing performance on the validation set, with a **fixed learning rate of 0.0001 used for all experiments** based on preliminary findings. The final, reported results for the best model configurations were then computed on the unseen test set to provide an unbiased estimate of generalization performance. Training was conducted for a fixed number of epochs (**20 for the Sequential Profiler and 5 for the GCN**) on an NVIDIA A100 GPU and NVIDIA H100 GPU.

*c) **Evaluation Metrics**:* Model performance was assessed using a suite of metrics appropriate for imbalanced classification.

- **AUPRC (Area Under the Precision-Recall Curve):** This was our primary metric for hyperparameter optimization and model comparison. Unlike AUROC,

AUPRC provides a more informative and conservative measure of performance on severely imbalanced datasets because it focuses on the trade-off between precision and recall for the rare positive class (illicit transactions). A higher AUPRC indicates a model that is better at identifying illicit transactions without being overwhelmed by false positives.

- **AUROC (Area Under the Receiver Operating Characteristic Curve):** We also report AUROC, a standard metric that measures a model's ability to discriminate between the two classes across all possible classification thresholds.
- **Precision, Recall, and F1-Score:** To evaluate a model's practical utility, we analyzed its performance at a specific decision threshold. This threshold was selected to maximize the F1-score on the validation set, representing a balanced operating point between precision and recall.
  - **Precision** measures the accuracy of the alerts (i.e., of all transactions flagged as illicit, what fraction are actually illicit?).
  - **Recall** measures the detection rate (i.e., of all existing illicit transactions, what fraction did the model find?).

This multi-faceted evaluation allows for both a robust comparison of the models' intrinsic ranking capabilities (AUPRC) and an assessment of their practical performance in a simulated deployment scenario.

IV. Experiments and Results

To evaluate and compare our two proposed architectures, the Dynamic Temporal GCN (Model A) and the End-to-End Sequential Profiler (Model B), we conducted a comprehensive suite of experiments. Our primary goal was to understand the sensitivity of each model's performance to its core hyperparameters and to determine which architectural philosophy is more effective for this task.

A. Experimental Setup

For both models, we employed a grid search methodology to explore their respective hyperparameter spaces. The primary evaluation metric for optimization was the Area Under the Precision-Recall Curve (AUPRC), which is well-suited for tasks with significant class imbalance, as is the case in our dataset. The performance of each experimental run was measured by the best AUPRC achieved on a held-out validation set.

B. Hyperparameter Analysis

The tuning process revealed distinct sensitivities for each model architecture. The detailed performance results for each experimental run are visualized in the dashboards presented in the Appendix.

*a) **Model A: Dynamic Temporal GCN**:* The performance of our graph-based model, summarized in the dashboard in Fig. 6 and Table V, was heavily influenced by parameters controlling the structure and depth of the local graph snapshots. The hyperparameter `Gnn_hidden_dim` was at a fixed value of 256.

- **K-History Size:** Defining the number of prior transactions to form a graph, this was tested with {20, 50, 100}. A clear positive correlation was observed, with **K=100** consistently yielding the best results, suggesting that a larger contextual graph is crucial for effective message passing.
- **GNN Layers:** The depth was tested for {2, 3, 5}. Performance improved with depth, with top-performing models all utilizing **5 layers**, indicating that multi-hop relational patterns are vital for this model.
- **Embedding Dimensions:** Varied across {32, 64, 128}, an account embedding dimension of **64** achieved the highest median and highest AUPRC scores, representing a balance between complexity of characteristics and risk of overfitting.

TABLE I
Best Hyperparameter Configuration and Final Performance for the Dynamic Temporal GCN (Model A).

Best Hyperparameter Configuration	
K-History Size	100
GNN Layers	5
Account Embedding Dim.	64
Transaction Emb. Dim.	64
Learning Rate	0.0001
Final Performance Metrics	
AUPRC	0.0235
AUROC	0.9276
Precision (at best F1)	18.1%
Recall (at best F1)	4.1%

*b) **Model B: End-to-End Sequential Profiler**:* The sequential model's performance was contingent on parameters governing the temporal sequence processing, summarized in the dashboard in Fig. 7 and Table IV. The hyperparameter `Gru_hidden_size` was at a fixed value of 128.

- **GRU Hidden Dimension:** Varied across {64, 128, 256}, performance improved with capacity. Both **128 and 256** dimensions produced significantly higher AUPRC scores than 64, suggesting a complex recurrent state is necessary to capture behavioral signatures.
- **Sequence Length:** Tested with {5, 10, 15}, there was a strong positive correlation with performance. A length of **15** consistently outperformed shorter lengths, indicating that a wider temporal context is critical.
- **Step Size:** Varied among {2, 5, 10}, a larger step size of **10** (less overlap between sequences) surprisingly yielded the best performance, likely by increasing training data diversity.

TABLE II
BEST HYPERPARAMETER CONFIGURATION AND FINAL PERFORMANCE FOR THE SEQUENTIAL PROFILER (MODEL B).

Best Hyperparameter Configuration	
Sequence Length	15
Step Size	10
GRU Hidden Dimension	256
Transaction Emb. Dim.	64 (fixed)
Learning Rate	0.0001
Final Performance Metrics	
AUPRC	0.1622
AUROC	0.9361
Precision (at best F1)	47.1%
Recall (at best F1)	20.9%

C. Comparative Performance and Thresholding

While AUPRC provides a holistic measure of ranking ability, practical deployment requires selecting a probability threshold to make classifications. We analyzed the precision-recall trade-off for both models at the threshold that maximized the F1-score.

A striking difference emerges when comparing the practical operating points of the two models. For the best-performing GCN model (Model A), a practical operating point yielded a **recall of 4.1% with a corresponding precision of 18.1%**. In contrast, the best Sequential models (Model B) demonstrated a vastly superior trade-off, achieving a recall of approximately **18-21%** with a high precision of **30-40%**. This indicates that Model B is not only better at ranking transactions but also operates at a much more effective and practical point for real-world application, identifying a larger portion of illicit activity with fewer false alarms.

D. Final Results and Model Comparison

The quantitative results from the hyperparameter search reveal a clear performance gap between the two architectural approaches. The overall distribution of AUPRC scores for Model A was concentrated between 0.015 and 0.02, with the best model reaching just over 0.023. In comparison, the scores for Model B were concentrated in a much higher range, between 0.12 and 0.14.

Synthesizing these findings, the best configuration for the **Dynamic Temporal GCN (Model A)** achieved a final AUPRC of **0.0235**. In contrast, the best configuration for the **End-to-End Sequential Profiler (Model B)** achieved a final AUPRC of approximately **0.16**, nearly a 7-fold improvement.

Overall, the End-to-End Sequential Profiler (Model B) not only achieved a dramatically higher AUPRC but also offered a more practical and effective precision-recall trade-off. To contextualize these results, we compare both models against our simple MLP baseline and the best-performing baseline published by the dataset's creators, Altman et al. (2023). Their strongest result on the LI-Small dataset was achieved using a Gradient Boosted Tree model with engineered graph features (GFP + XGBoost). As shown in Table III, our sequential profiler surpasses all baselines, demonstrating the power of learning behavioral signatures directly from temporal sequences over both localized graph structures and feature-engineered tree models.

TABLE III
FINAL PERFORMANCE COMPARISON ON THE TEST SET

Model	**AUPRC**	**AUROC**	**Prec.**	**Recall**	**F1**
MLP Baseline	0.0061	0.9005	0.3%	76.3% *	0.67%
Model A (GCN)	0.0235	0.9276	18.1%	4.1%	6.70%
Model B (Seq.)	**0.1622**	**0.9361**	**47.1%**	**20.9%**	**28.9%**
Baseline from Altman et al. (2023)					
GFP + XGBoost	-	-	-	-	27.3%

*Achieved with extremely low precision, resulting in a negligible F1-score.

V. DISCUSSION

Our comparative analysis provides a decisive result: the End-to-End Sequential Profiler (Model B) dramatically outperformed the Dynamic Temporal GCN (Model A). The failure of our MLP baseline (AUPRC 0.0061) confirms that context is essential, but our main findings suggest what kind of context matters most. The GCN's relatively poor performance stems from its localized, spatial receptive field. In contrast, the bidirectional GRU's success indicates that the long-range temporal dependencies within a single account's history contain a much richer predictive signal. This challenges the notion that graph structures are inherently optimal for all financial crime detection tasks.

A. Implications and Limitations

The superiority of the sequential profiler has significant practical implications. Model B's architecture aligns well with existing financial data paradigms (chronological ledgers), reducing the need for complex real-time graph construction. Furthermore, its superior precision-recall trade-off (20.9% recall at 47.1% precision vs. the GCN's 4.1% at 18.1%) translates to a higher detection rate with a more manageable volume of high-quality alerts for investigators.

However, this study has several key limitations. The use of a synthetic dataset means performance may not generalize to adversarial, real-world scenarios. Our hyperparameter search, while extensive, was not exhaustive, potentially leaving performance gains unrealized. The models also relied solely on transactional data, omitting richer information like KYC profiles. Finally, the sequential profiler lacks inherent interpretability, a significant barrier to regulatory adoption.

B. Future Work

Future work should prioritize validating these models on real-world financial data or extent its traning and testing on the larger dataset from Altman et al. (2023). A promising direction is a hybrid architecture where GRU-generated behavioral embeddings serve as initial node features for a global GNN, combining the strengths of both approaches. Integrating explainability techniques, such as attention mechanisms, is

crucial for providing actionable intelligence. Finally, employing more sophisticated methods like Bayesian optimization could more efficiently tune these models to their maximal potential.

VI. Conclusion

In this study, we conducted a rigorous comparative analysis of two prominent deep learning architectures for the task of money laundering detection on imbalanced data: a graph-based Dynamic Temporal GCN and an End-to-End Sequential Profiler. Our objective was to determine which modeling paradigm, capturing localized relational structures versus learning long-range temporal behaviors, is more effective for this complex problem. Our experimental results provide a clear and decisive answer. The End-to-End Sequential Profiler (Model B) dramatically outperformed the Dynamic Temporal GCN (Model A), achieving a final AUPRC of 0.1622 compared to 0.0235, a nearly seven-fold improvement. Furthermore, the sequential model offered a vastly superior and more practical precision-recall trade-off, identifying a larger fraction of illicit transactions with significantly fewer false alarms. The primary takeaway from this work is that for the patterns simulated in the AMLworld dataset, the temporal "signature" of an account's behavior contains a much richer and more predictive signal than its immediate, local graph neighborhood. This suggests that architectures like bidirectional GRUs, which are adept at capturing long-range dependencies and contextual nuances in sequential data, are exceptionally well-suited for this domain. Although graph-based methods remain powerful for many network-related tasks, our findings indicate that a sequence-first approach may be more fruitful for building effective next-generation AML systems. Acknowledging the limitation of using synthetic data, future work should focus on validating these findings on real-world transactional data and exploring hybrid models that combine the strengths of both sequential and graph-based reasoning.

References

Altman, E., Blanuša, J., Von Niederhäusern, L., Egressy, B., Anghel, A. & Atasu, K. (2023), Realistic synthetic financial transactions for anti-money laundering models, *in* 'Advances in Neural Information Processing Systems', Vol. 36, Curran Associates, Inc., pp. 29851–29874.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. & Bengio, Y. (2014), 'Learning phrase representations using rnn encoder-decoder for statistical machine translation', *arXiv preprint arXiv:1406.1078* .

Kazemi, S. M., Goel, R., Jain, K., Kobyzev, I., Sethi, A., Forsyth, P. & Poupart, P. (2020), 'Representation learning for dynamic graphs: A survey', *Journal of Machine Learning Research* **21**(70), 1–73.

Kingma, D. P. & Ba, J. (2014), 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980* .

Kipf, T. N. & Welling, M. (2016), 'Semi-supervised classification with graph convolutional networks', *arXiv preprint arXiv:1609.02907* . [Preprint]. Available at: https://arxiv.org/abs/1609.02907.

Kute, D. V., Pradhan, B., Shukla, N. & Alamri, A. (2021), 'Deep learning and explainable artificial intelligence techniques applied for detecting money laundering–a critical review', *IEEE Access* **9**, 82300–82317.

Ma, Y., Liu, X., Shah, N. & Tang, J. (2021), 'Is homophily a necessity for graph neural networks?', *arXiv preprint arXiv:2106.06134* . [Preprint]. Available at: https://arxiv.org/abs/2106.06134.

Mekruksavanich, S. & Jitpattanakul, A. (2022), 'Rnn-based deep learning for physical activity recognition using smartwatch sensors: A case study of simple and complex activity recognition', *Mathematical Biosciences and Engineering* **19**, 5671–5698.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. & Monfardini, G. (2009), 'The graph neural network model', *IEEE Transactions on Neural Networks* **20**(1), 61–80.

United Nations Office on Drugs and Crime (UNODC) (n.d.), 'Money-laundering: Overview'.
URL: *https://www.unodc.org/unodc/en/money-laundering/overview.html*

Weber, M., Domeniconi, G., Chen, J., Weidele, D. K. I., Bellei, C., Robinson, T. & Leiserson, C. E. (2019), Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics, *in* 'KDD Workshop on Anomaly Detection in Finance'. Anchorage, AK, USA, August.

Yu, Q., Xu, Z. & Ke, Z. (2024), Deep learning for cross-border transaction anomaly detection in anti-money laundering systems, *in* '2024 6th International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI)', IEEE, pp. 244–248.

Zhang, A., Lipton, Z. C., Li, M. & Smola, A. J. (2024), *Dive into Deep Learning*. [Online]. Accessed: 20 August 2025.
URL: *https://d2l.ai/chapter*$_r ecurrent-modern/gru.html$

Appendix

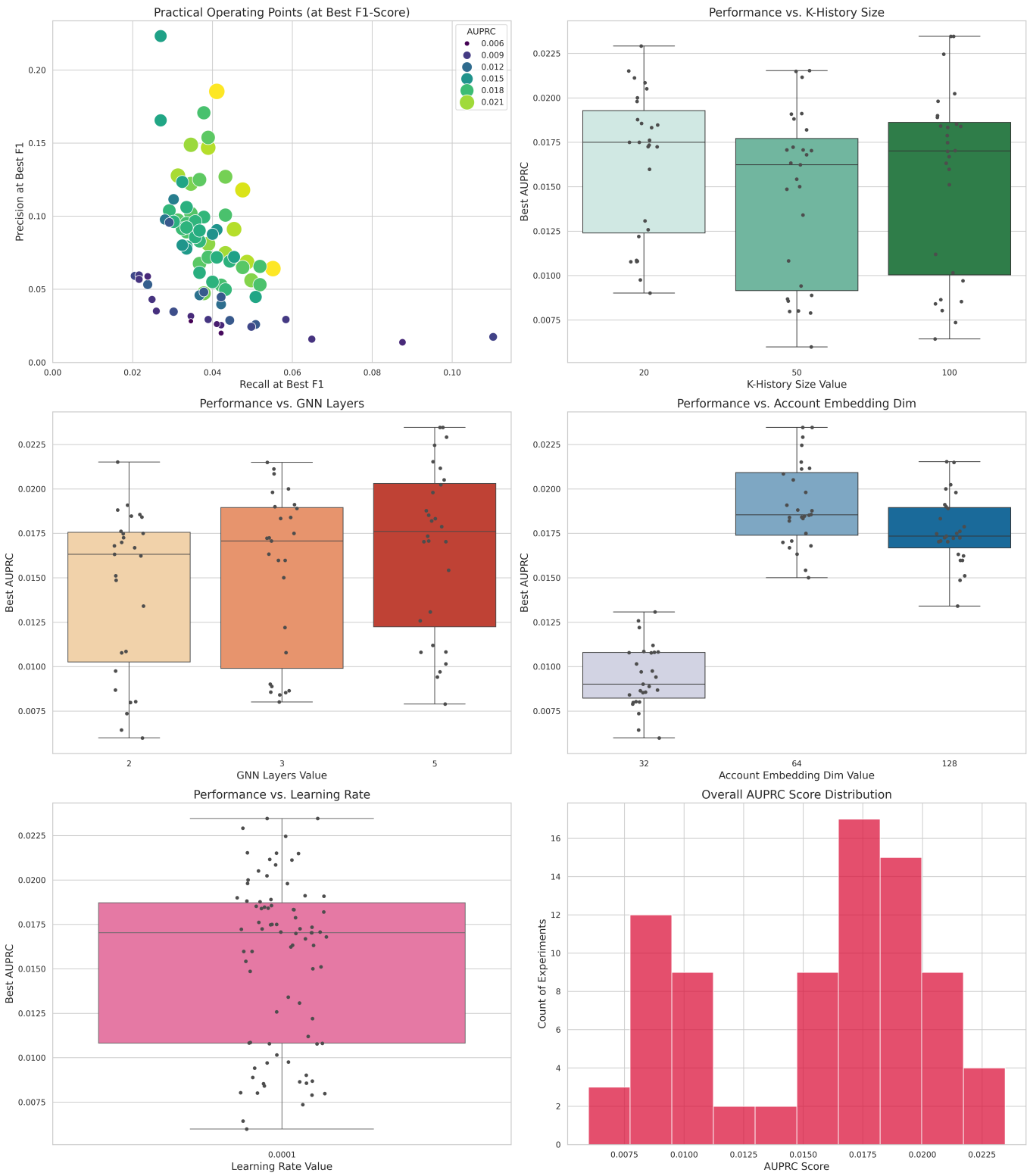


Fig. 6. Results from Model A

Fig. 7. Results from Model B

TABLE IV
DETAILED HYPERPARAMETER TUNING RESULTS FOR THE END-TO-END SEQUENTIAL PROFILER (MODEL B), SORTED BY AUPRC.

Tx. Emb. Dim.	Learning Rate	Seq. Length	Step Size	AUROC	AUPRC	Best F1	Threshold	Precision	Recall
256	0.0001	15	10	0.9361	0.1622	0.2893	0.9977	0.471	0.209
64	0.0001	15	10	0.9376	0.1580	0.2952	0.9960	0.440	0.222
128	0.0001	15	10	0.9349	0.1464	0.2717	0.9975	0.489	0.188
256	0.0001	10	2	0.9424	0.1362	0.2589	0.9980	0.360	0.202
128	0.0001	10	2	0.9391	0.1362	0.2556	0.9974	0.347	0.202
64	0.0001	10	2	0.9361	0.1334	0.2587	0.9958	0.343	0.208
128	0.0001	15	2	0.9375	0.1323	0.2518	0.9959	0.388	0.186
256	0.0001	15	2	0.9439	0.1306	0.2450	0.9974	0.365	0.184
128	0.0001	15	5	0.9310	0.1294	0.2448	0.9967	0.322	0.197
256	0.0001	15	5	0.9298	0.1289	0.2438	0.9970	0.435	0.169
64	0.0001	15	2	0.9346	0.1278	0.2396	0.9962	0.318	0.192
64	0.0001	15	5	0.9291	0.1261	0.2518	0.9960	0.454	0.174
128	0.0001	10	5	0.9324	0.1137	0.2376	0.9971	0.326	0.187
256	0.0001	10	10	0.9297	0.1136	0.2067	0.9977	0.337	0.149
256	0.0001	10	5	0.9375	0.1120	0.2369	0.9971	0.304	0.194
64	0.0001	10	5	0.9301	0.1097	0.2313	0.9948	0.265	0.205
128	0.0001	10	10	0.9286	0.1034	0.2127	0.9976	0.318	0.160
256	0.0001	5	2	0.9225	0.0903	0.2035	0.9970	0.231	0.182
64	0.0001	10	10	0.9255	0.0831	0.2000	0.9957	0.278	0.156
64	0.0001	5	2	0.9253	0.0759	0.1868	0.9929	0.186	0.188
128	0.0001	5	10	0.9173	0.0717	0.1582	0.9961	0.198	0.132
256	0.0001	5	5	0.9156	0.0672	0.1414	0.9979	0.197	0.110
128	0.0001	5	2	0.9265	0.0639	0.1755	0.9963	0.162	0.191
64	0.0001	5	10	0.9201	0.0580	0.1325	0.9981	0.308	0.084
64	0.0001	5	5	0.9205	0.0476	0.1282	0.9968	0.160	0.107
256	0.0001	5	10	0.9166	0.0436	0.1149	0.9976	0.180	0.084
128	0.0001	5	5	0.9182	0.0358	0.1266	0.9974	0.107	0.156

TABLE V
DETAILED HYPERPARAMETER TUNING RESULTS FOR THE DYNAMIC TEMPORAL GCN (MODEL A), SORTED BY AUPRC.

K-History	GNN Layers	Acc. Emb.	Tx. Emb.	Learning Rate	AUPRC	Best F1	Threshold	Precision	Recall
100	5	64	64	0.0001	0.0235	0.0670	0.9749	0.181	0.041
20	5	64	128	0.0001	0.0229	0.0528	0.9648	0.068	0.043
100	5	64	128	0.0001	0.0225	0.0669	0.9648	0.113	0.048
50	5	128	256	0.0001	0.0215	0.0587	0.9749	0.097	0.042
20	2	64	256	0.0001	0.0215	0.0545	0.9799	0.062	0.049
50	3	128	128	0.0001	0.0215	0.0597	0.9849	0.159	0.037
50	5	64	256	0.0001	0.0212	0.0454	0.9899	0.062	0.036
20	3	64	64	0.0001	0.0211	0.0526	0.9849	0.073	0.041
20	3	64	256	0.0001	0.0209	0.0553	0.9849	0.137	0.035
20	5	64	64	0.0001	0.0205	0.0485	0.9849	0.057	0.042
100	5	128	256	0.0001	0.0202	0.0512	0.9447	0.052	0.051
20	3	128	256	0.0001	0.0200	0.0433	0.9950	0.047	0.040
100	3	64	256	0.0001	0.0198	0.0620	0.9749	0.109	0.043
20	5	128	256	0.0001	0.0198	0.0414	0.9799	0.159	0.024
50	3	128	64	0.0001	0.0191	0.0556	0.9899	0.105	0.038
50	2	64	64	0.0001	0.0191	0.0601	0.9899	0.132	0.039
100	3	128	256	0.0001	0.0190	0.0457	0.9799	0.085	0.031
100	3	128	128	0.0001	0.0189	0.0521	0.9799	0.097	0.036
50	2	64	256	0.0001	0.0188	0.0472	0.9849	0.070	0.036
20	5	64	256	0.0001	0.0188	0.0446	0.9799	0.094	0.029
20	2	64	64	0.0001	0.0186	0.0308	0.9950	0.022	0.053
100	5	64	256	0.0001	0.0185	0.0515	0.9899	0.052	0.051
20	2	64	128	0.0001	0.0185	0.0602	0.9849	0.099	0.043
100	2	64	256	0.0001	0.0184	0.0492	0.9950	0.067	0.039
100	3	64	64	0.0001	0.0184	0.0600	0.9849	0.120	0.040
100	3	64	128	0.0001	0.0183	0.0563	0.9950	0.074	0.045
20	5	128	128	0.0001	0.0183	0.0431	0.9899	0.043	0.043
50	5	64	64	0.0001	0.0182	0.0471	0.9899	0.079	0.034
100	5	128	64	0.0001	0.0179	0.0443	0.9849	0.105	0.028
20	2	128	128	0.0001	0.0176	0.0445	0.9799	0.050	0.040
20	3	64	128	0.0001	0.0175	0.0530	0.9799	0.066	0.044
20	2	128	256	0.0001	0.0175	0.0361	0.9950	0.030	0.044
100	2	128	256	0.0001	0.0175	0.0495	0.9799	0.071	0.038
20	5	128	64	0.0001	0.0173	0.0499	0.9950	0.078	0.037
20	2	128	64	0.0001	0.0173	0.0454	0.9899	0.118	0.028
20	3	128	128	0.0001	0.0172	0.0395	0.9950	0.032	0.052
50	3	128	256	0.0001	0.0172	0.0523	0.9799	0.064	0.044
50	3	64	128	0.0001	0.0171	0.0429	0.9950	0.064	0.032
50	5	128	128	0.0001	0.0171	0.0504	0.9749	0.102	0.034
100	5	128	128	0.0001	0.0170	0.0441	0.9648	0.055	0.037
50	5	128	64	0.0001	0.0170	0.0490	0.9849	0.069	0.038
100	2	64	64	0.0001	0.0170	0.0426	0.9950	0.036	0.053
50	2	64	128	0.0001	0.0168	0.0467	0.9899	0.064	0.037
100	2	64	128	0.0001	0.0167	0.0449	0.9950	0.047	0.043
50	3	64	256	0.0001	0.0163	0.0332	0.9950	0.025	0.050
100	2	128	128	0.0001	0.0163	0.0469	0.9950	0.045	0.049
50	2	128	128	0.0001	0.0162	0.0480	0.9899	0.093	0.032
20	3	128	64	0.0001	0.0160	0.0471	0.9899	0.145	0.028
100	3	128	64	0.0001	0.0160	0.0539	0.9749	0.078	0.041
50	5	64	128	0.0001	0.0154	0.0450	0.9749	0.061	0.036
100	2	128	64	0.0001	0.0151	0.0367	0.9950	0.035	0.039
50	3	64	64	0.0001	0.0150	0.0538	0.9899	0.074	0.042
50	2	128	256	0.0001	0.0149	0.0516	0.9849	0.093	0.036
50	2	128	64	0.0001	0.0134	0.0461	0.9899	0.087	0.031
20	5	32	64	0.0001	0.0131	0.0406	0.9950	0.110	0.025
20	5	32	128	0.0001	0.0126	0.0391	0.9950	0.036	0.042
20	3	32	64	0.0001	0.0122	0.0390	0.9849	0.045	0.035
100	5	32	256	0.0001	0.0112	0.0317	0.9548	0.053	0.023
20	2	32	256	0.0001	0.0109	0.0411	0.9799	0.049	0.036
50	5	32	128	0.0001	0.0108	0.0384	0.9648	0.035	0.042
20	5	32	256	0.0001	0.0108	0.0347	0.9799	0.036	0.034
20	3	32	256	0.0001	0.0108	0.0300	0.9799	0.021	0.055
20	2	32	128	0.0001	0.0108	0.0338	0.9950	0.028	0.042
100	5	32	64	0.0001	0.0102	0.0302	0.9899	0.028	0.034
20	2	32	64	0.0001	0.0098	0.0276	0.9899	0.020	0.045
100	5	32	128	0.0001	0.0097	0.0306	0.9849	0.021	0.056
50	5	32	256	0.0001	0.0094	0.0280	0.9397	0.017	0.082