Optimizing Cutting Edge Machine Learning Models for Cryptocurrency Fraud Detection

Ari Asarch
Department of Psychiatry
University of Washington
aasarch@uw.edu

Jiachen Zhong
Department of Applied Mathematics
University of Washington
jczhong@uw.edu

William Tchen
Department of Statistics
University of Washington
wtchen1@uw.edu

Zongwan Cao
Department of Applied Mathematics
University of Washington
zongwanc@uw.edu

Introduction of Problem

The emergence of the cryptocurrency financial market poses significant challenges to financial security. This problem matters because it undermines financial stability and erodes public trust in this financial system. Thus, in this project, we evaluated a novel machine learning method for automatic detection of money laundering activities in cryptocurrencies. First, we began by recreating the novel work found in Vassallo, et al. (2021)[1]. Vassallo defined the features to be fed into the model as local and aggregated features and was attempting to predict whether the transaction was licit or illicit. Local features refer to specific details about the transaction itself, such as the timestep, number of inputs/outputs, and transaction fee. Aggregated features are derived from the transaction's neighboring data, including statistics like standard deviation, minimum, maximum, and correlation coefficients of neighboring transactions. However, the machine learning implementation by Vassallo theoretically failed to adequately capture time series data through both their segmentation of data and model choice, eXtreme Gradient Boosting (XGB). Thus, to determine if Vassallo's method and/or model can adequately incorporate time series data, we attempted to stack and evaluate a model that can adequately capture time series data, Long Short-Term Memory Neural Network (LSTM) on top of the model by Vassallo. Moreover, in case this stacking was inefficient at solving the time-series issue, we evaluated this model separately as well. It is important to mention that due to time constraints and virtual environment constraints, we built and performed hyperparameter tuning on an LSTM model using only the NumPy package with multiple layers, backpropagation, gradient clipping, dropout, and regularization, but recognize the limitations of our LSTM in comparison to more modern neural networks. Lastly, we then un-segmented the data to allow for time-series patterns to be retained and retested Vasallo's model, our stacked LSTM model, and un-stacked LSTM model. Therefore, in this paper we proposed solving the cryptocurrency fraud labeling issue proposed by Vassallo by using a model that can account for the time-series aspects of cryptocurrency transactions. However, since our model could not outperform Vassallo's models we suggest that further work would increase the hyperparameter search, optimization of the LSTM, and or the implementation of more modern neural networks that can handle time series data.

Problem Statement

Let there be I unique transactions in the dataset. Each transaction i is associated with a set of local features $\mathbf{l}_i \in \mathbb{R}^{m_l}$ and aggregated features $\mathbf{a}_i \in \mathbb{R}^{m_a}$. The target variable $y_i \in \{0, 1\}$ indicates whether the transaction is licit (0) or illicit (1). Each transaction also occurs at a specific timestep $t_i \in \mathbb{N}$ which is defined by month. Formally, the problem can be defined as follows: given the features $\mathbf{X}_i = \left[\mathbf{l}_i^T, \mathbf{a}_i^T, t_i\right]^T$ for each transaction i, the goal is to learn a function $f(\mathbf{X}_i)$ that predicts the target y_i: $\hat{y}_i = f(\mathbf{X}_i)$, where $\hat{y}_i$ is the predicted label for transaction i. The model $f(.)$ is trained to minimize the difference between the predicted labels $\hat{y}_i$ and the true labels y_i via the F1 score.

Preprint. Under review.

In consideration of the time series structure of the transaction data, the rolling window validation is employed. For the raw transaction dataset including T months, it will be divided into T partitions for each month denoted by $t_i \in [1, T]$. For the starting time point at month t_i, the machine will be trained on the training set including the transaction data with $t = t_i$, and will be tested on the test set involving the transaction data with $t = t_i + 1$. After storing the test results, the machine will roll to the next timepoint $t_i + 1$ and iterate the training and test process until the last partition has been tested. The model will be evaluated on the average of the stored test results from each iteration. The whole design of such a rolling window validation method is shown in Figure 1. To better illustrate the rolling window design of our study mathematically, where $t \in \{1, 2, \ldots, T-1\}$ and w is the window size, let: $\mathcal{T}_t = \{i \mid t_i = t\}$, $\mathcal{V}_t = \{i \mid t_i = t + 1\}$. The validation proceeds by iterating t from 1 to $T - 1$, training the model on $\mathcal{T}_t$ and evaluating on $\mathcal{V}_t$. The final validation metrics are obtained by averaging the performance metrics over all rolling windows: $\text{F1 Score}_{\text{avg}} = \frac{1}{T-1} \sum_{t=1}^{T-1} \text{F1 Score}(\mathcal{V}_t)$. This method helps smooth out short-term fluctuations and highlight longer-term trends, providing more stable parameter estimates and reducing the impact of outliers and random variability.

Review of Relevant Work

One of the earliest methods discovered to detect money laundering was a rule-based system that checked whether certain thresholds were numerically exceeded. Unfortunately, money laundering was detected by this system despite not actually being present in the dataset, leading to an incorrect assessment. To account for this, Senator et al. (1995) proposed the idea that tree-based machine learning models, such as Decision Trees, could be capable of detecting money laundering in finance related datasets [1] [2]. This proposal was revolutionary for the field of money laundering detection because there was a lack of sufficiently known models or algorithms to address this problem at the time. For the sake of brevity, various models and methods were used over the years to detect money laundering, but recently a lot of work has been focused on XGB models. As an example, Jullum et al. (2020) tested XGB on financial transaction related and found notable improvements to the old rule-based system previously mentioned and Decision Trees in terms of both reproducibility and computational efficiency [3]. Another example of XGB models used on financial datasets can be seen in Hajek et al. (2022) where the authors tested XGB on mobile payment systems and found that XGB to be well fitted to solve these issues due to its robust ability to handle class imbalance and data overfitting [4]. Due to the success of these newly tested tree-based models XGB seemed to be the leading alternative to detecting money laundering. However, with the rise of cryptocurrency, the utility of the models was yet to be tested. Cryptocurrency transaction data stands out from regular financial market data due to its use of cryptography to secure transactions. This cryptographic security increases the difficulty of detecting illicit activities, such as money laundering, because it makes the transactions harder to counterfeit [5]. Consequently, the complexity and richness of cryptocurrency transaction data offer numerous features that can be leveraged in machine learning models. These features can include transaction volumes, timestamps, wallet addresses, and the network structure of transactions, all of which can provide valuable insights for predictive analytics and anomaly detection in the financial domain. Moreover, this type of problem is a binary classification problem in that we are trying to predict whether a cryptocurrency transaction is fraud or not. Through the works of Marasi and Ferretti (2024), Lorenz, et al. (2020), and Contreras, et al. (2024), both supervised and unsupervised machine learning techniques have been researched and tested on cryptocurrency datasets depending on whether said dataset was sufficiently labeled or not [6] [7] [8]. Thus, due to the inherent differences in cryptocurrency markets from previously tested financial markets, the purpose of Vassallo, et al. (2021) was to investigate the strengths and weaknesses of various XGB versions to investigate its utility in cryptocurrency money laundering detection [1].

Though the data collection process Vassallo, et al. (2021), used is described in more detail along with the formal mathematical description of XGB and LSTMs in later sections, it is important for us to stress the idea that just because a model is performing well for a given dataset, that does not mean that it is theoretically the right model for that dataset. To expand, models like XGB are what's commonly referred to as model agnostic to time-series data, whereas LSTMs are not model agnostic to time-series data in that shuffling the indices of the data will not affect the outcome of the predictions from an XGB model but will drastically change the outcome of the predictions [9]. For example, suppose that a fraudster began with relatively small transactions, but then slowly increased the value associated with those transactions. Theoretically, an XGB model would not be able to even account for this time-series connection, whereas our LSTM model would be able to account for this. Additionally, as demonstrated by ConvLSTM (Convolutional LSTM), which is specifically designed

to process time-series financial transactions, we believe that this is a sound approach [10]. Therefore, due to the already shown success of Vassallo's model, in this project, we were interested in seeing if stacking an LSTM model to Vassallo's model or if our LSTM model were able to better capture cryptocurrency fraud.

Data Collection Process

To sufficiently retest the model, we used the same Elliptic dataset tested by Vassallo, et al. (2021) [1]. Specifically, the Elliptic dataset is composed of two smaller datasets called elliptic_txs_classes.csv and elliptic_txs_features.csv, which both needed to be pre-processed. A general description of the actual dataset is as follows: The dataset consists of 203,796 transactions, 4545 transactions were marked as illicit and 42,019 marked as licit with the remaining being undefined. Additionally, due to intellectual property issues, we cannot provide an exact description of all the features in the dataset. Preprocessing of the data set was as follows: First, the column transaction IDs in elliptic_txs_classes.csv were held to the values to be predicted and/or tested against in that 1) Illicit transactions were valued at 1, 2) Licit transactions that were valued at 2 and subsequently changed to 0, and 3) Unknown transactions simply discarded. In contrast, the pre-processing of elliptic_txs_features.csv did not follow a specific procedure but rather involved condensing the data in a more straightforward manner. For instance, the 167 columns in this dataset were manually labeled as follows: 1) The first two columns were uniquely named "txId" and "ts" indicating transaction ID and timestep respectively, 2) The following 93 columns in this dataset represented local features where each of these columns was named "LF_#" where # corresponded to the number of the column disregarding the first two columns, and 3) The remaining 72 columns represented aggregated features instead and were thus named "AF_#" in a similar manner to the previous section. It is important to mention that Local Features refer to local information about the transaction including the timestep, number of inputs/outputs and transaction fee. Aggregated Features obtained using transaction information one-hop backward/forward from the center node, such as standard deviation, minimum, maximum and correlation coefficients of neighboring transactions, along with the same information extracted for the local features (for example, transaction fee and inputs/outputs). Before processing, the CSV file did not include headers and so the features needed to be manually labeled: Additionally, some transactions labeled as illicit and licit in the elliptic_txs_features.csv file can be seen below:

Displaying 8 of 167 columns:

txId	ts	LF_1	LF_2	LF_3	LF_4	...	AF_71	AF_72
339095110	13	-0.169145	-0.158782	-1.201368	-0.121969	...	-0.120613	-0.119792

Table 1: Licit transaction

txId	ts	LF_1	LF_2	LF_3	LF_4	...	AF_71	AF_72
232438397	13	0.163054	1.963789	-0.646376	12.409294	...	-0.120613	-0.119792

Table 2: Illicit transaction

Additionally, we then utilized several data sampling techniques to alleviate the issue of class imbalance, since there are much more licit transactions than illicit transactions, which are as follows: 1) By using SMOTE, which stands for Synthetic Minority Over-Sampling Technique, we randomly oversampled the data of the less populated classes. 2) However, since this sampling technique had the primary drawback of leading to overfitting due to oversampling of a specific class, we utilized NCL (Negative Correlation Learning). For reference, NCL randomly under-samples units of the more populated classes, but, as a consequence, results in general information loss. 3) Thus, we then applied a combination of these two sampling techniques called NCL-SMOTE to undersample the more populated classes (Licit transactions) and oversample the less populated classes (Illicit transactions).

Lastly, the authors performed sub-sectioning of the data into intervals which is described as follows: The rolling window time series validation method is adopted in our model. An example of the work flow is presented below to show the design of the validation method. The time series data of bitcoin transactions is divided into individual timesteps by months, and the validation is proceeded by rolling the training and test set over the timesteps. By averaging the stored statistics of all the rolling windows, the final validation metrics are obtained. Such rolling window time series validation method can be helpful in smoothing out the short-term fluctuations of the time series and highlighting

longer term trends. For instance, this method can provide more stable estimates of parameters but also reduce the impact of outliers and random variability, offering more generality of the final model.

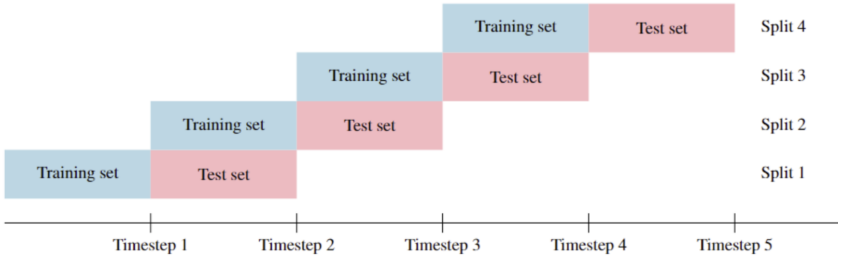

Figure 1: Rolling Window

However, one of the downsides of this sub-sectioning is losing granularity within the data between the smoothed timesteps. For example, if a series of illicit transactions was performed in sequence from one month to another, then this rolling window method would miss this. Therefore, we ran an experiment to see if the absence of this rolling window could increase model performance by capturing more granularity in the data by splitting the data with a 70/30 split seen below.

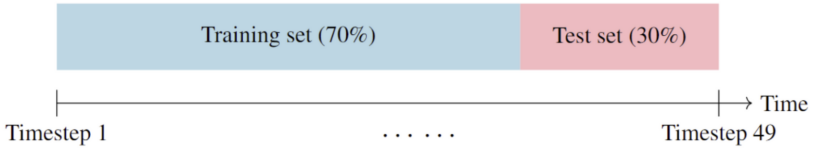

Figure 2: No Rolling Window i.e. Entire Time Series

Mathematical Background

XGBoost

The XGBoost stands for Extreme Gradient Boosting, the fastest and best-integrated decision tree algorithm that was proposed by Chen, et.al (2016) [2]. This model is fundamental for the novel stacked gradient-boosted model we are going to introduce and is heavily used as the base model of that generalized model. The XGBoost can be described from the aspects of the objective function and interactive method[3].

Objective function: Consider a dataset $D = \{(x_i, y_i) : i = 1, \ldots, n, x_i \in \mathbb{R}^m, y_i \in \mathbb{R}\}$, where n is the number of observations, each observation consisting of m features, and each has a corresponding output y. The output y_i for each observation is generated by an ensemble method, described by a generalized model: $y_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i)$ (1). Here, f_k denotes a regression tree, and $f_k(x_i)$ is the score provided by the k-th tree for the i-th data point. The ensemble aims to optimize the following regularized objective function: $\mathcal{L}(\phi) = \sum_{i=1}^{n} L(y_i, \phi(x_i)) + \sum_{k=1}^{K} \Omega(f_k)$ (2). In equation (2), L is a loss function that measures the discrepancy between the predicted and actual values. To control the complexity of the model and mitigate overfitting, a regularization term Ω is included, defined as: $\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \|w\|^2$ (3). The parameters γ and λ are regularization coefficients that penalize the number of leaves T and the magnitude of the leaf weights w, respectively. This regularization helps to simplify the model and enhance its generalizability by reducing overfitting.

Interactive method: The objective function is minimized through an interactive method. For example, the objective function that is minimized in j-th iterative is: $\mathcal{J} = \sum_{i=1}^{n} L(y_i, y_i^{(j-1)} + f_j(x_i)) + \Omega(f_j)$ (4) which can be further simplified with Taylor expansion. Loss reduction can be achieved by Splitting a decision node into two child nodes in a decision tree. The loss reduction formula can be expressed as follows: $\Delta L = \sum_{i \in I} L(i) - \left(\sum_{j \in \text{left}} L(j) + \sum_{k \in I_{\text{right}}} L(k) \right)$ (5). Where I is the set of all observations in the current node before the split, I_{left} and I_{right} are the subsets of observations in the left and right child nodes, respectively, after the split, and $L(i)$ represents the loss associated with observation i. The functions $L(I_{\text{left}})$ and $L(I_{\text{right}})$ are defined as the aggregated losses for the subsets

of observations in the left and right nodes, respectively: $L(I_{\text{left}}) = \sum_{j \in I_{\text{left}}} L(j), \quad L(I_{\text{right}}) = \sum_{k \in I_{\text{right}}} L(k)$ (6). The expression ΔL measures the decrease in loss from a split, aiding in choosing the best way to divide observations at each node to minimize the model's total loss.

XGBoost excels over other tree-boosting methods due to several key features: **Regularization**: XGBoost incorporates both L1 and L2 regularization in its loss function, which helps prevent overfitting more effectively than traditional boosting methods that may lack regularization. **Scaling of Tree Contributions**: Each tree's influence on the final model can be scaled down, reducing overfitting risks and leading to a more stable ensemble. **Column Sampling**: By randomly selecting features for each tree—similar to techniques in random forests—XGBoost increases diversity and robustness against overfitting. These enhancements make XGBoost a highly effective tool for creating precise and reliable predictive models.

LSTM

Long short-term memory (LSTM) is a state-of-the-art technique for sequence learning that was first proposed by [6]. The LSTM networks belong to the class of recurrent neural networks (RNN), with the advantage of understanding the long-term dependency of time series, selective memory, and better control of gradient flow[7]. The figure 2 illustrates a simple LSTM memory cell. For the LSTM networks used in this paper, the mathematical details of the composite functions are presented as follows: **Forget gate** $\mathbf{f}_t$: $\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f)$, **Input gate** $\mathbf{i}_t$: $\mathbf{i}_t = \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i)$, **Output gate** $\mathbf{o}_t$: $\mathbf{o}_t = \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)$, **Cell input activation** $\tilde{\mathbf{c}}_t$: $\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c)$. **Cell state** $\mathbf{c}_t$: $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$, **Hidden state** $\mathbf{h}_t$: $\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$. **Output** $\mathbf{y}_t$: $\mathbf{y}_t = \mathbf{W}_y \cdot \mathbf{h}_t + \mathbf{b}_y$. In the equations above, σ denotes the logistic sigmoid function, which is used for the gate activations. The variables $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{o}_t$, and $\tilde{\mathbf{c}}_t$ respectively represent the input gate, forget gate, output gate, and cell input activation vectors at time t. These vectors, as well as the hidden state $\mathbf{h}_t$ and cell state $\mathbf{c}_t$, are all of the same dimensionality as the hidden vector $\mathbf{h}$. The weight matrices $\mathbf{W}_i$, $\mathbf{W}_f$, $\mathbf{W}_o$, and $\mathbf{W}_c$ denote the consolidated weights that connect both the input vector $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$ to each respective gate and the cell input. Similarly, $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_c$, and $\mathbf{b}_o$ are the bias terms for the input gate, forget gate, cell input activation, and output gate, respectively. Each weight matrix and bias vector is tailored to modulate the flow of information through the network, ensuring that the LSTM can learn complex temporal dynamics effectively.

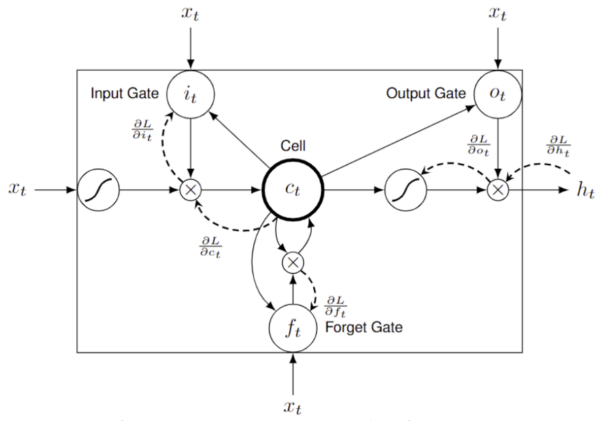

Figure 3: Structure of LSTM memory cell with forward and backward paths

The backward propagation of the LSTM involves computing the gradients of the loss function L to update the model's parameters. The gradients are calculated using the chain rule at the function of $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \text{next}} \times \frac{\partial \text{next}}{\partial x}$ for each component like h_t, o_t, i_t, and f_t. Backward passing these gradients is very important for the modeling training, allowing the LSTM to adjust its weights and improve accuracy by minimizing the loss over training iterations.

Description of Algorithm (ASXGB)

Initially, we decided to rerun the exact algorithm featured in Vassallo, et al. (2021), which was coined as Adaptive Stacked eXtreme Gradient Boosting (ASXGB) by the authors [1]. The algorithm begins with H, an XGBoost model or classifier, that is trained on Dk, which is a dataset produced through the predictions of N base models labeled h1,..., hN (also XGBoost classifiers). Specifically, for each

round k, H is trained on Dk and examines the features of this dataset and measures each feature's importance. Afterward, for each Dk, all of h1,..., hN generate 2N binary features which are added to the previously examined features, producing a measurement of the performance, hgain, of each h as a classifier. Then the total number of rounds where hgain is taken from each h, represented as hrounds, is measured. Finally, for each h, if hrounds is greater than or equal to nrounds, the compiler computes the summation of the previous hgain for nrounds. If the hrounds is greater than or equal to nrounds condition is satisfied for an h, then this h is placed into a pool of h's to be replaced. The h with the smallest value of the summation of the previous hgain for nrounds is replaced by a new XGBoost model and then the algorithm loops back to the beginning. The algorithm map can be found below:

Algorithm 1 Adaptive Stacked Extreme Gradient Boosting (ASXGB)

Global Variables: $winSize$, $winBufferX$, $winBufferY$, $nBaseModels$, $metaModelTrainRatio$
Initialize buffers and model counts
function ADAPTIVETRAIN(X, y)
    Store data in buffers: $winBufferX$, $winBufferY$
    if buffer size $\geq winSize$ **then**
        $batchX$, $batchY \leftarrow$ Extract batch from buffers
        Train or update base models with $batchX$, $batchY$
        Update ensemble using UPDATEENSEMBLE
        Replace weakest model using GETWEAKESTBASEMODELINDEX
    end if
end function
function UPDATEENSEMBLE
    for each base model **do**
        Predict and update meta-features
        Adjust weights and performance metrics
    end for
end function
function PREDICT(X)
    if not all base models trained **then**
        return majority vote from base models
    else
        Aggregate base model predictions
        return meta model prediction
    end if
end function

Moreover, we used the two main variations of the most successful and previously described model, ASXGB, were compared by pooling the F-score for each time iteration: ASXGB and ARF. A brief description of each is as follows: 1. As previously explained, Adaptive eXtreme Gradient Boosting (ASXGB) is an enhanced XGBoost algorithm designed to handle concept drift in data streams by incorporating mechanisms that update the model ensemble dynamically as new data arrives. 2. Adaptive Random Forest (ARF) is an extension of the Random Forest algorithm that, similar to ASXGB, is designed to handle concept drift in data streams by using a combination of bagging and random sub-spacing, along with an ensemble of decision trees where it can phase out less accurate or outdated trees.

Description of General Difficulties with the Problem

We encountered a multitude of issues during this phase of the project. Due to the scarcity of information surrounding machine learning methods in the context of money laundering, we had difficulty interpreting the algorithm used in Vassallo, et al. (2021) with the intention of modifying it eventually [1]. To elaborate on this point, due to the lack of ReadMe's or general description in the original source GitHub for this algorithm, we needed to infer the function of specific sections of the algorithm at times. Furthermore, it was challenging to manage the virtual environment containing the algorithm due to the presence of several processing concerns. In addition, attempting to match the original paper's data format with new datasets that we have acquired has proven to be more difficult than expected. Since we plan to rerun the algorithm in Vassallo, et al. (2021) on other datasets eventually, it is imperative that we match the original paper's data format to ensure consistency in the

algorithm's output. Moreover, recreating a functional LSTM with multiple layers, backpropagation, gradient clipping, dropout, and regularization using only NumPy was more difficult than previously thought.

Experimental Layout

As a broad overview, we executed the following plan: 1) Figure out the dependency issues associated with Vassallo, et al. (2021) [1] as there were no ReadMe or installation instructions associated with the source code. 2) Establish that we could recreate the results found in Vassallo, et al. (2021) [1]. 3) Build an LSTM model with multiple layers, backpropagation, gradient clipping, dropout, and regularization using only numpy and attach it onto the existing best model and perform hyperparameter tuning. 4) Rerun the four total models with with and without the rolling window design regarding timesteps. 5) Lastly, we reran the models with some feature engineering i.e. whether to include the aggregated features or not since AF includes both the Local and Aggregated Features. It is important to mention that if given more time, we would attempt to compare the LSTM created with different models more apt for time series rather than XGB and/or Random Forest (RF), but due to the complexity associated with creating the LSTM, we decided that the best alternative was to compare it to the existing model. Thus, future work would include a more robust model testing set and a recreation of Vassallo's model with a more modern framework. A broad overview of the experiments we then ran can be seen below:

Features Used	**Timesteps**	**Models Ran**			
LF	Rolling Window	ASXGB	LSTM	LSTM + ASXGB	ARF
LF	Entire Time Series	ASXGB	LSTM	LSTM + ASXGB	ARF
AF	Rolling Window	ASXGB	LSTM	LSTM + ASXGB	ARF
AF	Entire Time Series	ASXGB	LSTM	LSTM + ASXGB	ARF

Table 3: Models Ran with Different Features and Timesteps Design

Results and Findings

Hyperparameter Tuning: After performing hyperparameter tuning on the base LSTM model i.e. the one not stacked onto the ASXGB model we found with regards to the F1 score our final hyperparameters of 32, 20, 0.2, 0.000001, 0.1, and 1.1 for units, epochs, dropout, learning rate, weight decay, and binary cross entropy scale factor respectively. We performed the hyperparameter search this way instead of a grid search due to the runtime associated with a single run, and thus, opted instead to iteratively update our model based on each hyperparameter tuning. 1) A small decrease in performance by increasing the number of units from 32 to 64 to 128 with results of 0.30, 0.25, 0.15 respectively. 2) A general increase in performance by increasing the number of epochs up to 20 ran from 5, 10, 20, and 40 with results of 0.11, 0.25, 0.33. 0.33 respectively. 3) A decrease in performance from increasing dropout to 0.2, 0.3, 0.4, and 0.5 with results of 0.33, 0.24, 0.33, 0.24 respectively. 4) An increase in performance by decreasing the learning rate from 0.0001, 0.00001, to 0.000001 with results of 0.21, 0.14, 0.30 respectively. 5) An increase in performance by increasing the weight decay from 0.1, 0.01, and 0.001 with results of 0.35, 0.34, 0.33 respectively. 6) Lastly, an increase in performance by decreasing the binary cross entropy scale factor from 0.9, 1.0, and 1.1 with results of 0.30, 0.29, 0.27 respectively. Thus, after performing the hyperparameter search we then ran the following experiments described in more detail in the Experimental Layout section and the F1 scores are as follows:

AF - Rolling Window: We found 0.27 (+/- 0.31), 0.22 (+/- 0.22), 0.69 (+/- 0.30), and 0.64 (+/- 0.38) respectively for the LSTM-ASXGB model, LSTM model, ASXGB model, and the ARF model (Figure 4). Thus, it can be seen that the ASXGB outperformed all other models, but was relatively close to the ARF model. Additionally there was a marked increase in performance when comparing the LSTM-ASXGB model to the LSTM model, but still, these models were performing nearly as half as well as the other models. Moreover, the results from each time step can be seen in Figure 5, where we see that both the LSTM-ASXGB model and LSTM model struggled with class imbalance towards the end where the LSTM-ASXGB had less consistent performance when compared to the LSTM model that performed quite well as it learned the time-series connection, but rapidly failed towards the end. Additionally, the ARF seemed to struggle more with the beginning and end of the timesteps when compared to the ASXGB model.

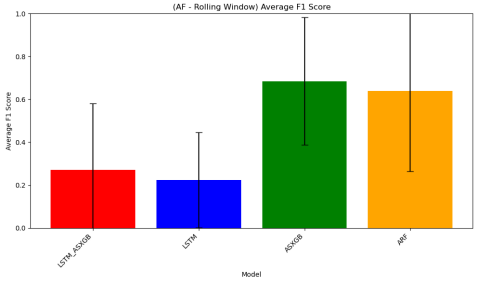

Figure 4: F1 score Evaluation of AF - Rolling Window

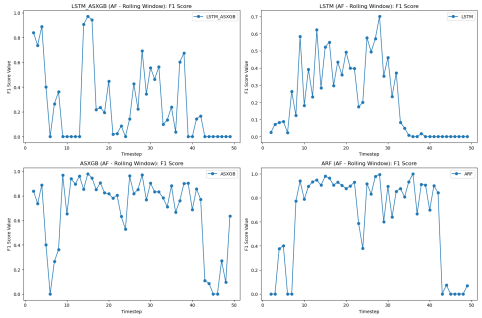

Figure 5: F1 score Evaluation of AF - Rolling Window Over Timesteps: Stacked LSTM (Top Left), LSTM (Top Right), ASXGB (Bottom Left), ARF (Bottom Right)

LF - Rolling Window: We found 0.31 (+/- 0.33), 0.21 (+/- 0.21), 0.68 (+/- 0.33), 0.66 (+/- 0.37) respectively for the LSTM-ASXGB model, LSTM model, ASXGB model, and the ARF model (Figure 6). Thus, it can be seen again that the ASXGB outperformed all other models, but with a slight decrease in performance and was still relatively close to the ARF model. Additionally, there was again a marked increase in performance when comparing the LSTM-ASXGB model to the LSTM model, but still these models were performing nearly as half as well as the other models. These results indicate that the AF features include more predictive features for all models except for the LSTM model that performed slightly worse. Moreover, the results from each time step can be seen in Figure 7 where we see a near similar performance pattern when compared to the AF F1 scores for each model over the rolling window.

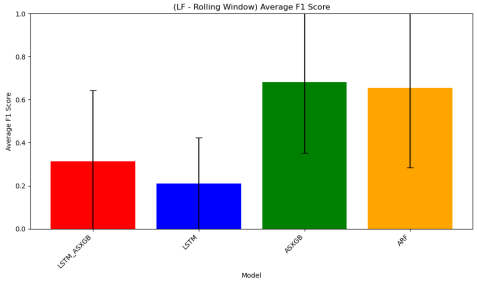

Figure 6: F1 score Evaluation of LF - Rolling Window

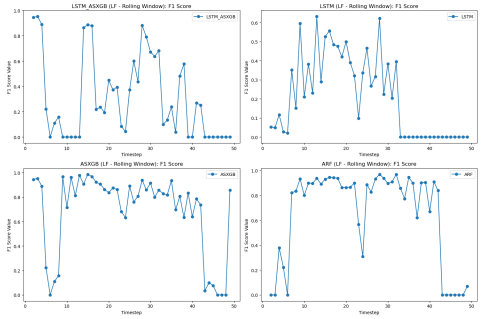

Figure 7: F1 score Evaluation of LF - Rolling Window Over Timesteps: Stacked LSTM (Top Left), LSTM (Top Right), ASXGB (Bottom Left), ARF (Bottom Right)

AF - Entire Time Series: We found 0.12, 0.03, 0.67, and 0.77 respectively for the LSTM-ASXGB model, LSTM model, ASXGB model, and the ARF model (Figure 8). Thus, it seems that the lack of a rolling window moderately decreased the ASXGB model's performance by 2%, but greatly increased the performance of the ARF model by 10%. Moreover, both the LSTM-ASXGB model and the LSTM model performed remarkably worse than the rolling window.

LF - Entire Time series: we found 0.12, 0.03, 0.72, and 0.78 respectively for the LSTM-ASXGB model, LSTM model, ASXGB model, and the ARF model (Figure 9). Interestingly, it seems that the lack of AF features in the data set when using the entire time series increases the performance of the ASXGB and ARF models, but not the LSTM-ASXGB model and LSTM model.

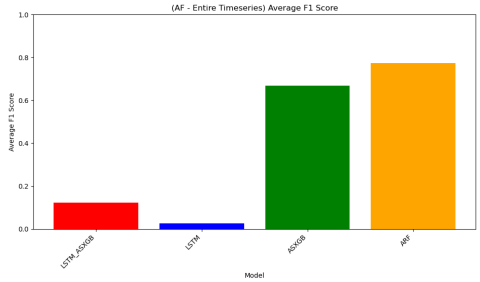

Figure 8: F1 score Evaluation of AF - Entire Time Series

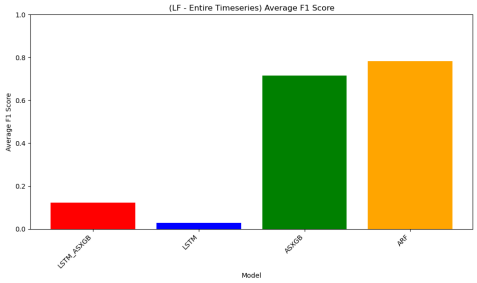

Figure 9: F1 score Evaluation of LF - Entire Time Series

Discussion

The results from our study offer valuable insights into the efficacy of different modeling approaches for detecting illicit activities in cryptocurrency transactions. Our findings highlight the strengths and limitations of various models, particularly in how they handle different feature sets (AF vs. LF) and temporal splits (rolling window vs. entire time series).

After performing hyperparameter tuning on the base LSTM model, i.e., the one not stacked onto the ASXGB model, we hypothesize that these were determined to be the best hyperparameters due to the following reasons: **Units**: A small decrease in performance was observed by increasing the number of units suggestsing that while increasing the model complexity can often capture more intricate patterns, in our case, it likely led to overfitting and/or an inability to generalize well. **Epochs**: A general increase in performance was noted by increasing the number of epochs up to 20 indicating that training for a sufficient number of epochs was crucial for the model to learn effectively, but beyond 20 epochs, there was not much improvement. **Dropout**: A decrease in performance was observed from increasing dropout past 0.2. Since dropout helps in regularization by preventing overfitting, a too high dropout rate might lead to underfitting in that the model fails to learn enough from the data. Thus, it seemed like the sweet spot for this was 0.2. **Learning Rate**: An increase in performance was seen by decreasing the learning rate allowing the model to converge more gradually by avoiding the risk of overshooting the optimal weights, and for a dataset with rapid changes in classes, this seemed to be the most effective. **Weight Decay**: Performance increased by increasing the weight decay since weight decay acts as an additional regularizer by penalizing large weights reducing overfitting, and thus, a higher weight decay likely provided a better balance of weights for our model. **Binary Cross-Entropy Scale Factor**: Lastly, an increase in performance was found from decreasing the binary cross-entropy scale factor to 0.9. Adjusting this scale factor likely helped in balancing the loss calculation more effectively, ensuring that the model does not overly penalize certain predictions. Thus we are confident in these hyperparameters as they were carefully chosen to balance the complexity and regularization of the model, enabling our LSTM model to learn as best as possible from the data without overfitting.

As demonstrated, the aggregated features (AF) generally outperformed the local features (LF) across most models indicating that the aggregated features, which capture neighborhood information around each transaction, are more informative for detecting illicit activities compared to using local transaction features alone. For instance, Illicit activities often involve coordinated patterns across multiple transactions, which local features may miss. However, for the ASXGB and ARF models on the entire time series split, using just the LF led to a slight increase in performance suggesting that in some cases, when models can effectively capture global patterns over the full data sequence, the local features may encode sufficient information. This indicates that the AF within the rolling window may be more effective at encoding global patterns missed by the LF on a smaller scale that turn into a detriment when all the LF are available rendering the AF moot. However, using a rolling window approach generally led to better performance across all models compared to training on the entire time series at once likely because the rolling window allows models to better adapt to dynamic patterns and concept drift in the data over time. Additionally, the entire time series split with a 70/30 train/test split may have suffered from a significant class imbalance issue in the test set due to a decrease in positive instances for testing towards the end of the sequence, leading to the poor performance of the LSTM-based models. For instance, the rapid decrease in performance of the LSTM model can be seen in Figure 10 below.

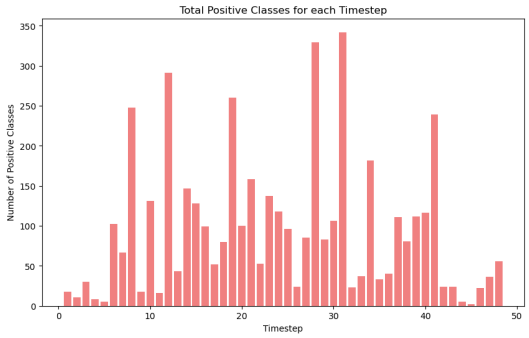

Figure 10: Total Positive Classes for each Timestep

As for models, the ASXGB and ARF models had comparable performance on the rolling window experiments, with ASXGB having a slight edge. However, on the entire time series split, the ARF outperformed ASXGB. This could be due to the fact that the ARF model is more robust to distributional shifts and class imbalances in the data when compared to an XGB model. With regards to our LSTM, the LSTM-ASXGB model consistently outperformed the base LSTM model by a large margin, demonstrating the benefit of combining the LSTM with the ASXGB ensemble for capturing both temporal patterns (LSTM) and non-linear feature interactions (ASXGB). We hypothesize that this is due to the LSTM being able to model long-range dependencies over the transaction sequences, while the ASXGB captures complex decision boundaries in the feature space. However, since we weighted both models equally, this may have led to a shift towards the LSTM's poor performance and a loss of the complex decision boundaries over space. Moreover, even though we performed NCL/SMOTE on the data for handling class imbalance, the LSTM alone struggled with this task regardless of the time series split or AF/LF. We hypothesize that the NCL-SMOTE technique was inefficient at solving the issue of LSTMs being difficult to train on highly imbalanced sequential data with long-range dependencies where ensemble tree-based methods are known to handle imbalanced data better, making the NCL/SMOTE technique sufficient for these models. Thus, further experiments would warrant other ways to include either the LSTM in conjunction with the ASXGB model and strategies to improve robustness to class imbalances and distributional shifts in sequential data.

Overall, our findings underscore the importance of considering both feature aggregation and appropriate temporal splitting strategies in designing models for cryptocurrency fraud detection. For instance, we have highlighted the effectiveness of a rolling window for cryptocurrency fraud transactions in that since the field is ever changing with new techniques or fraud instances, a rolling window is better suited to handle these minute changes. However, it is important to mention that defining this rolling window accurately may provide some challenges as it might not adequately capture changes at the right time and warrants further investigation. Thus, future work should explore more sophisticated ensemble techniques and hybrid models that can leverage the strengths of both sequential and tree-based methods, experimentation with rolling window definitions, along with advanced data balancing techniques to better handle the challenges posed by imbalanced datasets.

Group Member Contributions

Ari Asarch: Figured out the environmental dependencies necessary to run the code without a ReadMe or any instructions and wrote a ReadMe and created a .yaml file for ease of use. Wrote the LSTM base model using only the NumPy package with multiple layers, backpropagation, gradient clipping, dropout, and regularization and integrated into the existing codebase and stacked it on top of the ASXGB model. Added predict and eval_proba functions to all models for metric analysis. Wrote the Discussion, Results and Findings, and Experimental Layout sections. **Jiachen Zhong**: Wrote the Formal Description of Algorithm Used, Mathematical Background, and Problem Definition sections. Further optimized the existing LSTM to decrease the runtime speed for a single run. Double checked LSTM instantiation and codebase. **Zongwan Cao**: Discovered what and how the data collection process occurred and recreated it. Wrote the Data Collection section. Designed, implemented, ran and analyzed the hyperparameter tuning experiments. Double checked LSTM instantiation and codebase. **William Tchen**: Wrote the Description of General Difficulties with the Problem, Review of Relevant Work, and Introduction of Problem sections. Analyzed results and created the plots for the hyperparameter tuning.

References

[1] Dylan Vassallo, Vincent Vella, and Joshua Ellul. Application of gradient boosting algorithms for anti-money laundering in cryptocurrencies. *SN Computer Science*, 2(3):143, 2021.

[2] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[3] Dahai Zhang, Liyang Qian, Baijin Mao, Can Huang, Bin Huang, and Yulin Si. A data-driven design for fault detection of wind turbines using random forests and xgboost. *IEEE Access*, 6: 21020–21031, 2018. doi: 10.1109/ACCESS.2018.2818678.

[4] Steven Farrugia, Joshua Ellul, and George Azzopardi. Detection of illicit accounts over the ethereum blockchain. *Expert Systems with Applications*, 150:113318, 2020.

[5] Rodrigo Colnago Contreras, Vitor Trevelin Xavier da Silva, Igor Trevelin Xavier da Silva, Monique Simplicio Viana, Francisco Lledo dos Santos, Rodrigo Bruno Zanin, Erico Fernandes Oliveira Martins, and Rodrigo Capobianco Guido. Genetic algorithm for feature selection applied to financial time series monotonicity prediction: Experimental cases in cryptocurrencies and brazilian assets. *Entropy*, 26(3):177, 2024.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[7] Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2017.11.054. URL `https://www.sciencedirect.com/science/article/pii/S0377221717310652`.

[8] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[9] Ted E Senator, Henry G Goldberg, Jerry Wooton, Matthew A Cottini, AF Umar Khan, Christina D Klinger, Winston M Llamas, Michael P Marrone, and Raphael WH Wong. Financial crimes enforcement network ai system (fais) identifying potential money laundering from reports of large cash transactions. *AI magazine*, 16(4):21–21, 1995.

[10] Joana Lorenz, Maria Inês Silva, David Aparício, João Tiago Ascensão, and Pedro Bizarro. Machine learning methods to detect money laundering in the bitcoin blockchain in the presence of label scarcity. In *Proceedings of the first ACM international conference on AI in finance*, pages 1–8, 2020.