Securing Ethereum: Machine Learning & Deep Learning Approaches to Smart Contracts and Transaction Security

1. INTRODUCTION

Smart contracts and Ethereum have significantly impacted the field of decentralized applications and blockchain technology. In 2022, the estimated value of the worldwide smart contracts market was approximately USD 1750 million. Projections suggest a substantial increase to roughly USD 9850 million by the year 2030, with an expected compound annual growth rate (CAGR) of about 24% from 2023 to 2030.

Ethereum is the backbone of the decentralized finance movement, offering financial instruments without the need for traditional financial intermediaries, through the use of smart contracts. Ethereum introduced the concept of smart contracts to the blockchain. These are self-executing contracts with the terms directly written into code, which execute automatically when conditions are met, without the need for a middleman. Finally, Ethereum's smart contracts and tokens are the foundational technology for the emerging metaverse, allowing users to create, buy, sell, and trade virtual goods and services securely and transparently.

1.1 Ether and ERC20

Ether (ETH) is the native cryptocurrency of the Ethereum blockchain platform. Among cryptocurrencies, Ether is second only to Bitcoin in market capitalization. Ether is used as the "fuel" to execute smart contracts on the Ethereum network. When someone wants to execute a smart contract, they must pay a transaction fee in Ether.

Ether does not inherently require a smart contract for basic transactions. However, smart contracts become involved when you want to execute more complex operations with Ether, such as automated distributions, interactions within decentralized applications, or other programmable actions that are beyond simple transfers.

ERC20 tokens inherently rely on smart contracts. The ERC20 standard itself is a set of rules and functions defined in a smart contract. Actions like transferring ERC20 tokens, checking balances, or approving tokens for use in other contracts are all functions defined in the ERC20 token's smart contract.

In summary, while basic Ether transactions can occur without the direct involvement of smart contracts, any complex operations with Ether, as well as all actions with ERC20 tokens, require interacting with smart contracts on the Ethereum blockchain. Smart contracts provide programmable functionality that allows both Ether and ERC20 tokens to be used in a wide range of decentralized and automated applications.

1.2 Smart Contracts

Smart contracts offer enhanced security and transparency due to their inherent characteristics and the way blockchain technology operates. They have an immutable code, they are decentralized and programmable logic.

However, while smart contracts themselves offer these security and transparency features, they are only as secure as the code they are written in. Poorly written or audited smart contracts can contain vulnerabilities that may be exploited, which has happened in several high-profile cases in the blockchain space. Therefore, this work offers crucial tools for ensuring the security and reliability of smart contracts.

The objective of this project is to predict when an ether transaction is fraudulent and when a smart contract is not safe.

2. DATA OVERVIEW

We used two different datasets to tackle 2 common issues in the Ethereum World:

1. As a usual transactions made over Ethereum, how can we know based solely on Ether and ERC20 information if the transaction is fraudulent or not?
2. What if the fraudulent behavior relies solely on the smart contract rather than the Ether and ERC20 information?

To solve these questions we used the following datasets for each problem respectively:

1. **Ether Transactions:** Kaggle Ether Fraud Detection: https://www.kaggle.com/code/nitinchan/ether-fraud-detection/input
2. **Smart Contracts:** mwritescode/slither-audited-smart-contracts: https://huggingface.co/datasets/mwritescode/slither-audited-smart-contracts

3. METHODOLOGIES

3.1 Ether transactions

Data Preprocessing

We started with a database with 51 columns and 9841 rows. After data analysis and cleaning we ended up with a final dataset of 17 columns (including the fraud flag). We removed columns that didn't incorporate any different information to the transactions (variance of zero) and variables with high pairwise correlation.

The dataset exhibited imbalance, prompting the utilization of undersampling techniques to address this issue. Undersampling aimed to alleviate the imbalance by reducing the dominance of the majority class, potentially enhancing the model's capacity to discern patterns within the minority class, particularly in instances of fraudulent transactions. The final training dataset contained 50% fraudulent transactions and 50% not fraudulent transactions.

Additionally, the dataset contained four columns with missing data exceeding 25%. To handle this, the OptKNN Imputation Learner was employed to both fit and transform the training set. This method facilitated the imputation of missing values within these columns, ensuring a more complete and robust dataset for subsequent modeling and analysis.

3.2 Smart Contracts

Data Preprocessing

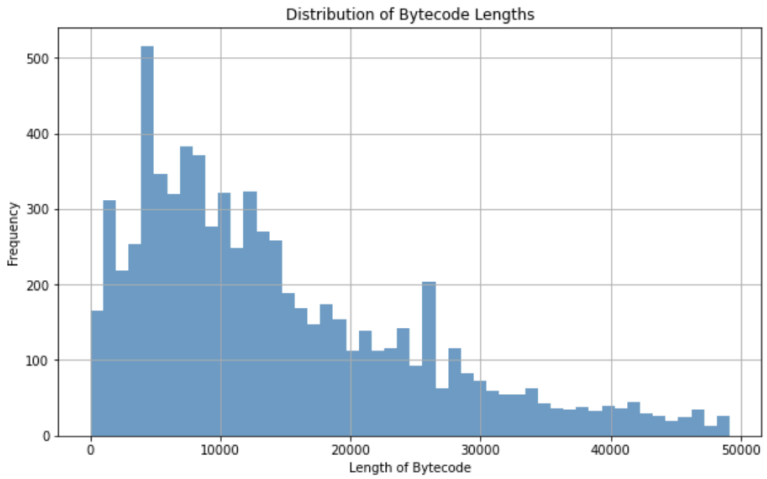
The original dataset consisted of 4 columns in total:

- address: a string representing the address of the smart contract deployed on the Ethereum main net
- source_code: a flattened version of the smart contract codebase in solidity
- bytecode: a string representing the smart contract's bytecode obtained when calling web3.eth.getCode(). Note that in some cases where this was not available, the string was simply 0x (so we had missing values).
- slither: list of class labels

For pre-porcessing the data:

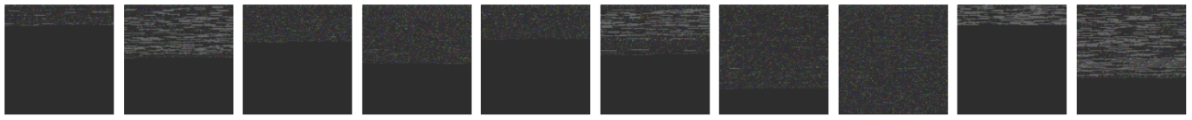
- The bytecode length was calculated
- The target label was simplified into safe or not safe. The original dataset had a multi-label classification but it was decided to adopt a binary approach similar to the Ether approach that was taken.
- Duplicates were removed
- Bytecodes with length 0 were removed. No imputation method was considered as the structure of the bytecode was very special.
- Bytecode was converted to string

- The dataset was unbalanced (34% safe vs 66% not safe) so we downsampled to the minority class.
- Splits were 60%, 25% and 15% for Train, Test and Validation respectively.
- Lastly, even though the majority of the bytecode lengths seem to be between 5K to 15K we decided to pad with zeros every bytecode until it reached the maximum possible length: 49,152.



- Lastly, the smart contract bytecodes were transformed into 128x128 RGB images using PIL.Image.frombytes() and stored into folder paths: training, validation and testing using ImageDataGenerator.

  Sample Image from the training set:



Models:

Exploring various strategies with our datasets presented unique challenges based on the characteristics of the data. On one side, smart contracts data had too little information to create models with trees so the approach was to convert the codebytes into images and creates a sequential CNN to train the bytecodes. On the other hand, we were able to perform optimal classification trees for the ethern transactions data.

- Ethern Transactional data

The tabular data that we obtained was used to to predict wheter the ethern transaction was fraud or not via 4 different types of models learned in class: CART, Random Forest, XGBoost and OCT.
Using IAI package in Julia, we ran several iterations and a cross validation with different maximum depths from 1 to 7 and min buckets from 10 to 30 for the different models.

- Smart contracts

A Sequential CNN model was used to train the bytecode images of the smart contract. We ran several configurations for the CNN and in the results section, we show the 3 models with the highest performance measured via accuracy and AUC. Overall, we noticed that incrementing the number of epochs leads to a worse performance which makes sense as a very large number of epochs can lead to overfitting. Furthermore, we also saw the model yielded on average better results when using the optimizer Adam as opposed to SGD like SGD(learning_rate=0.001, momentum=0.8, nesterov=True). This is also intuitive as the Adaptive Momentum (Adam) is a combination of Adadelta (which recursively computes the sum of past gradients using exponential smoothing) and momentum (which accelerates SGD in the relevant direction).

An early-stopping approach was implemented to avoid overfitting. This technique involves stopping the training once the model performance stops improving on a held-out validation dataset.

Finally, we also used batch normalization to fit the model with a batch_size of 64, which can help in generalizing the model by reducing internal covariate shifts. We compiled our models using the accuracy metric.

4. RESULTS

4.1 Ethern Transactional data

The initial application of classification tree methodologies resulted in deep trees, reaching a depth of 7. Despite achieving AUC and accuracy metrics surpassing 98%, the interpretability of these trees was compromised due to their complexity. Consequently, an adjustment was made to limit the potential tree depths to 3.

This refinement preserved a high level of accuracy exceeding 90% in predicting fraudulent transactions based on their attributes. By reducing the tree depth, we balanced the model's interpretability while maintaining a commendable level of predictive accuracy.

	CART	**Random Forest**	**XGBoost**	**OCT**
AUC	0.907	0.959	0.991	0.908
Accuraccy	0.918	0.954	0.968	0.932

Although the XGBoost and Random Forest had higher levels of accuracy, the OCT was much more interpretable. In this last case, the most relevant variables where the difference between the first and last transaction, number of Unique ERC20 tokens received, number of ERC20

token transactions sent to Unique account addresses, the average value sent, the number of sent transactions, the total ERC sent in the contract. Find in the appendix 2 the tree.

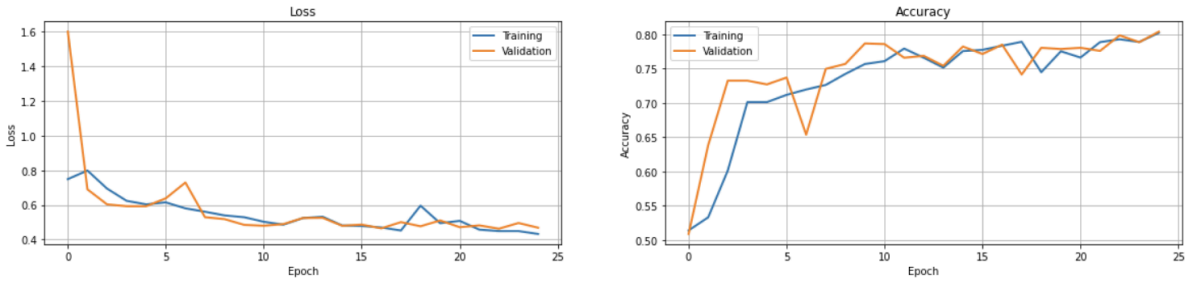
4.2 Smart contracts

As mentioned in a previous section, we ran several configurations for our CNN. However, we showcase the configurations used for the 3 models with the highest performance:

	CNN1	**CNN2**	**CNN3**
Layers*	24	20	24
Kernel Size	3,1 for Conv2D	3,1 for Conv2D	3,1 for Conv2D
L2 regularization	1e-5	1e-4	1e-5
Epochs	25	20	25
Total Parameters	1,004,945	4,015,137	1,004,945
Optimizer	SGD(learning_rate= 0.001, momentum=0.8, nesterov=True)	adam	adam
Loss	binary_crossentropy	binary_crossentropy	binary_crossentropy
Val Accuracy	0.7441	0.7803	0.8038
Train Accuracy	0.7310	0.7638	0.8021
Test Accuracy	0.7353	0.7792	0.8091

*takes into account the input, output and Flatten layer.

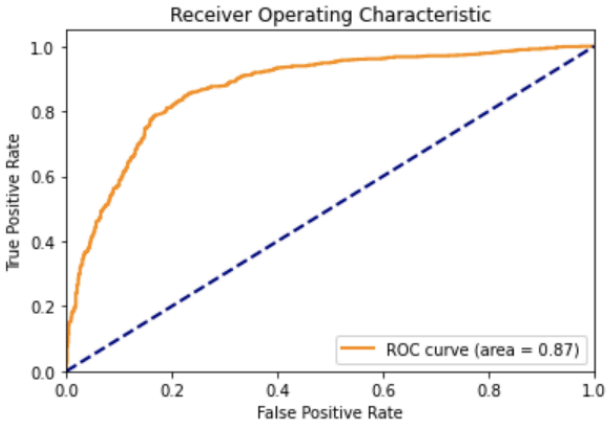
Based on these results, the best model was the CNN3 with an accuracy out of sample of 80.91%, meaning that the model correctly predicted the outcome 80.91% of the time across all classes. Finally, it yielded an AUC of 87.07%.



Loss and Accuracy plots for CNN3

From the loss plot chart, we notice the training and validation loss decrease sharply in the first few epochs, which indicates that the model is learning from the data. After the initial drop, both losses fluctuate but tend to follow a similar trend, which suggests the model is not overfitting significantly. The validation loss does not consistently exceed the training loss, which is positive as it suggests the model generalizes well.

From the accuracy plot chart, we can see the training accuracy improves quickly, which is expected as the model begins to fit the training data. As for the validation accuracy, this increases alongside the training accuracy, indicating that the model is generalizing well to the unseen validation data.
After around epoch 5, the validation accuracy exceeds the training accuracy, which is unusual. It could be due to having a small validation set, certain regularization techniques that impact training more than validation, or simply variability in the data. However, after epoch 20 both validation and training data seem to converge nicely. Overall, the model appears to be performing well in terms of not overfitting, as indicated by the convergence of training and validation metrics.



AUC plot for CNN3

5. CONCLUSION AND NEXT STEPS

In this work, we adopt the learnings from the course and especially put into action the philosophy of tailoring a solution to a specific problem, meaning tackling a problem with different perspectives leveraging the knowledge of models we have at our disposal. A problem can be seen from different perspectives and therefore it requires different approaches. In the new world of blockchain, we sometimes don't get the complete picture of the data and we need to understand how to handle it. In this case, we realized that features of the transactions such as amount sent, time difference, etc, are good predictors for fraudulent transactions. But in the case we cannot count with this information we can use deep learning with smart contracts.

In the next steps and future research when data becomes available, we propose an embedded multimodal model to join both ethern transactions and smart contracts to create a more robust model to detect fraud.

6. CONTRIBUTIONS

We both contributed from the meticulous pre-processing of the data for both datasets, which is the bedrock of any robust model, to the diligent and iterative process of model training for the Ether and Smart Contracts tasks. Our collaborative approach ensured that the tasks of cleaning, organizing, and transforming the data into a workable format were shared equally, reflecting our team's cohesive work ethic and dedication to creating a reliable foundation for our models.

Furthermore, as we progressed to the subsequent phases of model selection, training, and refinement, we shared an equal responsibility. Each of us actively participated in running various models, rigorously tuning hyperparameters, and meticulously evaluating model performance. This was especially helpful for the CNN models as they took between 40 minutes to 50 minutes to run each. Furthermore, the task of scrutinizing the accuracy and ensuring the models' generalizability was a joint endeavor.

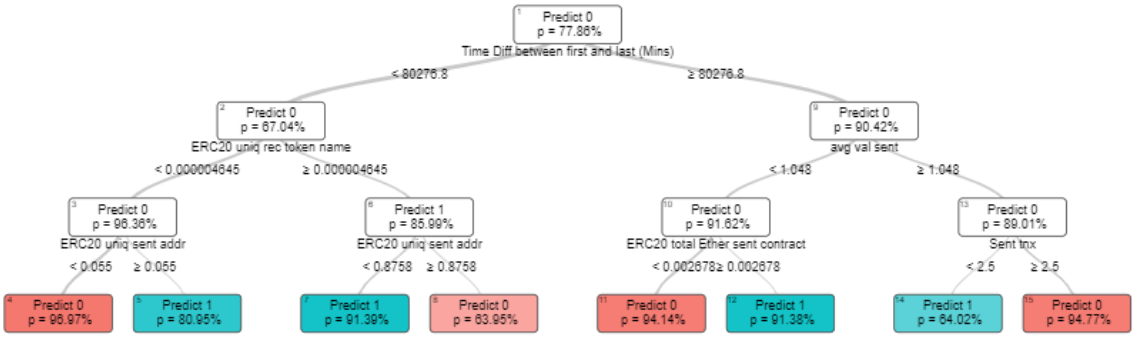
Finally, for this written report we divided the task of writing our findings in the Ether model and Smart Contract model. The same was extended to the creation of the presentation.

Appendix 1

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 128, 128, 16)      448

 conv2d_1 (Conv2D)           (None, 128, 128, 16)      272

 leaky_re_lu (LeakyReLU)     (None, 128, 128, 16)      0

 max_pooling2d (MaxPooling2D  (None, 64, 64, 16)       0
 )

 conv2d_2 (Conv2D)           (None, 64, 64, 32)        4640

 conv2d_3 (Conv2D)           (None, 64, 64, 32)        1056

 leaky_re_lu_1 (LeakyReLU)   (None, 64, 64, 32)        0

 max_pooling2d_1 (MaxPooling  (None, 32, 32, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 32, 32, 64)        18496

 conv2d_5 (Conv2D)           (None, 32, 32, 64)        4160

 leaky_re_lu_2 (LeakyReLU)   (None, 32, 32, 64)        0

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 conv2d_6 (Conv2D)           (None, 16, 16, 128)       73856

 conv2d_7 (Conv2D)           (None, 16, 16, 128)       16512

 leaky_re_lu_3 (LeakyReLU)   (None, 16, 16, 128)       0

 max_pooling2d_3 (MaxPooling  (None, 8, 8, 128)        0
 2D)

 conv2d_8 (Conv2D)           (None, 8, 8, 256)         295168

 conv2d_9 (Conv2D)           (None, 8, 8, 256)         65792

 leaky_re_lu_4 (LeakyReLU)   (None, 8, 8, 256)         0

 max_pooling2d_4 (MaxPooling  (None, 4, 4, 256)        0
 2D)

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 128)               524416

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 1,004,945
Trainable params: 1,004,945
Non-trainable params: 0
_________________________________________________________________
```

Complete Configuration for the highest CNN performance (CNN3)

Appendix 2


1 Predict 0
p = 77.86%
Time Diff between first and last (Mins)
< 80276.8
≥ 80276.8
2 Predict 0
p = 67.04%
ERC20 uniq rec token name
< 0.000004645
≥ 0.000004645
9 Predict 0
p = 90.42%
avg val sent
< 1.048
≥ 1.048
3 Predict 0
p = 96.36%
ERC20 uniq sent addr
< 0.055
≥ 0.055
6 Predict 1
p = 85.99%
ERC20 uniq sent addr
< 0.8758
≥ 0.8758
10 Predict 0
p = 91.62%
ERC20 total Ether sent contract
< 0.002678
≥ 0.002678
13 Predict 0
p = 89.01%
Sent tnx
< 2.5
≥ 2.5
4 Predict 0
p = 96.97%
5 Predict 1
p = 80.95%
7 Predict 1
p = 91.39%
8 Predict 0
p = 63.95%
11 Predict 0
p = 94.14%
12 Predict 1
p = 91.38%
14 Predict 1
p = 64.02%
15 Predict 0
p = 94.77%