AUTOMATED DETECTION AND CATEGORISATION OF CRYPTOCURRENCY RUG PULL SCAMS

A dissertation submitted for

CO880: Project and Dissertation

as part of the program: MSc Advanced Computer Science

supervised by

Dr Darren Hurley-Smith

By

Napatchol Thaipanich

September 2025

Word Count:

10072

Abstract

Rug pull scams are among the most damaging forms of fraud in decentralised finance (DeFi), with global losses exceeding $27 billion. These scams erode trust in blockchain ecosystems, complicate regulatory oversight, and undermine investor confidence. Existing detection tools such as CRPWarner, TrapdoorAnalyser, and RPHunter show the feasibility of automated analysis, yet each suffers from critical limitations, including reliance on scarce verified source code, narrow coverage, and high computational costs. These gaps highlight the need for scalable approaches that adapt to evolving attack strategies.

This dissertation investigates whether rug pull scams can be detected and categorised automatically using transaction and bytecode data, even without source code. A framework was designed that integrates supervised classification, anomaly detection, and semi-supervised self-learning. The system fuses multiple feature modalities—transaction statistics, opcode sequences, and sequential behaviours—within a late-fusion ensemble whose weights are optimised using Optuna. Self-learning expands labelled datasets with high-confidence pseudo-labels, addressing data scarcity and enabling adaptability to new scam categories.

The framework was evaluated on CRPWarner, RPHunter, and Trapdoor datasets. On CRPWarner ground truth, the system achieved strong performance (macro F1 $\approx$ 0.91). On RPHunter, results were more moderate (macro F1 $\approx$ 0.56), reflecting severe class imbalance. On a 2% Trapdoor sample, the framework generalised effectively (macro F1 $\approx$ 0.94). The anomaly detection module achieved perfect recall, acting as a fail-safe for unseen behaviours. These findings confirm that rug pulls leave detectable blockchain patterns and that multi-source fusion improves robustness. Contributions include a reproducible framework, a pseudo-labelling mechanism for dataset expansion, and insights into runtime trade-offs for deployment.

Acknowledgements

I would like to sincerely thank my supervisor, **Dr. Darren Hurley-Smith**, for his invaluable ideas, guidance, and support throughout this project. His insights and direction shaped the development of this work and were essential in bringing it to completion. I am also grateful to the **University of Kent, School of Computing**, for preparing the academic foundation that enabled this research and for providing the resources and environment necessary to carry it out.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Decentralised Finance (DeFi) is a peer-to-peer financial system built on public blockchains and cryptocurrencies, allowing individuals to transfer funds directly to one another without intermediaries such as banks and other financial service providers. There are concerns about vulnerabilities in the deployed code on the chain, known as smart contracts, as well as the potential for hacks and scams. [4] Therefore, the responsibility lies with the user who aims to use this channel to make a transaction.

Among the most damaging are *rug pull* scams, where promoters attract investment into a project or liquidity pool and subsequently remove, redirect, or manipulate the assets to their benefit, leaving investors with illiquid or worthless tokens [1]. Beyond immediate monetary losses, estimated at over $27 billion globally [5], these scams erode trust in blockchain ecosystems and complicate risk assessment for regulators, platforms, and users. Therefore, this dissertation addresses the challenge of detecting and categorising rug pull scams automatically.

1.1 Objectives

The main objective of this project is to design, implement, and evaluate a machine learning system that can automatically detect and categorise rug pull scams. Specifically, the dissertation seeks to:

- Investigate existing rug pull detection tools and techniques, including static analysis, opcode features, transaction graph modelling, and deep learning. [6]

- Explore whether detection can be achieved using only transaction data and compiled bytecode, given that less than 1% of contracts are publicly verified.
- Integrate self-learning mechanisms that allow the model to adapt to new scams and expand its labelled dataset over time.
- Evaluate the performance of the system against benchmark datasets and compare results with existing tools such as CRPWarner, TrapdoorAnalyser, and RPHunter.

This dissertation is guided by the following central research question: *Can rug pull scams be automatically detected and categorised using blockchain transaction data—even when source code or bytecode is unavailable—through a scalable, self-learning machine learning system that adapts to emerging scams and supports cross-chain deployment?*

1.2 Structure of the Dissertation

The dissertation is organised into five chapters. Following Chapter 1, which introduces the background, research problem, and objectives, Chapter 2 reviews the existing literature, beginning with the broader context of financial scams before narrowing to cryptocurrency rug pulls. It also surveys relevant detection techniques, ranging from logic-based analysis and opcode features to graph and machine learning approaches, and critically compares the strengths and limitations of prominent tools such as CRPWarner, TrapdoorAnalyser, and RPHunter. Chapter 3 describes the methodology adopted for this research, including the identification of gaps in prior work, the datasets employed, the design of the detection framework, and the self-learning strategy proposed to address data scarcity. Chapter 4 presents the implementation and experimental evaluation of the system, reporting results across multiple datasets and comparing performance with existing benchmarks. Finally, Chapter 5 concludes the dissertation by summarising the key findings, reflecting on limitations, and future work. Together, these chapters provide a structured progression from problem definition to solution design, evaluation, and reflection.

Chapter 2

Literature Review

This chapter reviews the rug pull scam landscape, including definitions, key characteristics, and notable cases. It then examines existing detection approaches, highlighting current tools alongside their comparative strengths and limitations. In addition, the chapter explores machine learning and deep learning techniques that support automated detection, including feature engineering, anomaly detection, and optimisation methods such as hyperparameter tuning. These insights provide the foundation for developing a proof-of-concept (PoC) detection tool in the subsequent chapters.

2.1 Concept of Digital Assets

Digital assets are cryptographic representations of value or rights recorded on a distributed ledger. A public blockchain maintains an append-only ledger secured by consensus [7]. On programmable networks such as Ethereum, *smart contracts* are on-chain programs that hold assets and enforce rules automatically [8]. Two broad categories are common: (i) *native cryptocurrencies* (e.g., ETH), which are integral to the protocol; and (ii) *tokens* issued by smart contracts. Fungible tokens typically implement the ERC-20 interface (e.g., `transfer`, `approve`) to ensure wallet and exchange interoperability [9], while non-fungible tokens (NFTs) often follow ERC-721 for unique digital items [10]. Users control assets via wallets (public addresses) secured by private keys; transfers settle directly on-chain once included in a block.

Tokens typically define tokenomics (mint/burn rights, fees/taxes, allocation, vesting)

prior to listing. On DEXs like Uniswap, AMMs set prices algorithmically and LPs who deposit token pairs receive LP tokens denoting their share [11, 12].

For example, Tether USD (USDT) is the largest U.S. dollar–pegged stablecoin by market capitalisation. It is designed to trade near $1.00 via primary redemptions with the issuer and secondary-market arbitrage, and it claims 1:1 backing by reserves (largely cash and short-dated U.S. Treasuries) with regular attestations by BDO Italia [13, 14]. As of August 2025, USDT's market capitalisation is approximately $167 billion, making it the dominant stablecoin by a wide margin [15, 16]. Tether Ltd. is part of the iFinex corporate group associated with the Bitfinex exchange; the firm has faced U.S. regulatory actions related to past disclosures about reserves [17, 18]. Despite controversy, USDT is widely used for liquidity and settlement across centralised and decentralised markets.

2.2 Rug Pull Scam Landscape

2.2.1 Definition and Characteristics

The issue of financial scams in the digital asset space is a growing concern, as fraudsters exploit the anonymity, speed, and largely unregulated nature of this ecosystem. Unlike traditional financial systems, which benefit from oversight by banks and regulatory bodies, cryptocurrency markets often operate with minimal safeguards, making them particularly vulnerable to abuse. Common fraudulent strategies include creating a token and generating hype to attract investors, only for the perpetrators to abruptly withdraw all liquidity, leaving investors with worthless tokens. Another tactic involves artificially inflating the value of a cryptocurrency through misleading information and coordinated trading; once the price peaks and retail investors have entered, the fraudsters liquidate their holdings, causing the value to collapse and resulting in substantial investor losses. [19]

A rug pull is one of the most common forms of scam in the Decentralised Finance (DeFi) ecosystem. In a simple rug pull, a malicious actor deploys a token on the blockchain and creates a trading pool by pairing their crafted token with a valuable asset, such as wrapped Ether (wETH), at a chosen ratio. Unsuspecting investors are

lured into the pool and purchase the scam tokens with their wETH or other legitimate assets.

Once a certain threshold of investment is reached, the malicious actor often sells or withdraws all tokens from the pool by redeeming and burning all liquidity provider (LP) tokens. As a result, investors are unable to swap their holdings back for valuable assets, and the purchased scam tokens become essentially valueless [1].

Assessment tools, such as the proof-of-concept detector developed in this dissertation and research systems like CRPWarner and RPHunter, aim to surface contract and market risk factors (e.g., tokenomics, code-level trapdoors, and transaction-flow anomalies) so that prospective investors can better appraise the risks of a specific token and avoid scam projects [3,20]. Most current solutions operate off-chain and therefore rely on users to actively consult them; consequently, their effectiveness depends on investor awareness and engagement. Complementing code analysis, transaction-behaviour profiling provides independent signals about a token's lifecycle and market dynamics—such as rapid liquidity withdrawals, concentrated control of LP tokens, or wash-trade-like patterns—which can alert users to elevated risk. In this light, the proposed PoC tool is positioned both as a detection aid and as an investor-education instrument, presenting salient tokenomics (supply, distribution, liquidity locks, and fee/tax rules) alongside behaviour-based indicators to support informed decision-making [1].

Identification Characteristics

Key features that differentiate rug pull scams from legitimate trading pools include [1]:

- Most malicious tokens originate from decentralised exchanges (DEXs).
- They disproportionately target inexperienced investors.
- The time between token creation and pool deployment is typically very short.
- No additional liquidity providers, apart from the creator, usually contribute to the pool.
- LP tokens from malicious pools are rarely locked.

- Minting events (creation of new tokens) are heavily clustered at the start of the pool's lifecycle.

2.2.2 Types of Rug Pulls [1]

Prior studies classify rug pulls into several types, as outlined below:

- **Simple Rug Pull**: The sole liquidity provider removes all tokens from the trading pool, leaving investors unable to exchange their holdings for valuable assets.
- **Sell Rug Pull**: The scammer either withholds a surplus of scam tokens at launch or mints more later. These tokens are then swapped for valuable assets (e.g., wETH), draining the pool.
- **Smart Contract (SC) Trapdoors**: Malicious functions are embedded within the token's smart contract, enabling the creation of unlimited tokens, forced transfers, or destruction of tokens held by investors. For example:
  - **Hidden Mint Function**: A function that allows developers to generate any number of tokens to any address.
  - **Limiting Sell Order**: A logic to restrict users from selling tokens
  - **Leaking Token**: A logic to leak tokens from other users without permission
- **Liquidity Pool (LP) Manipulation**: The scammer manipulates liquidity conditions, e.g., by locking pools, conducting wash trades to simulate activity, or executing pump-and-dump schemes to inflate prices.
- **Counterfeit Token**: Tokens mimic legitimate ones or falsely claim affiliation with well-known brands, deceiving investors into believing they are authentic.
- **Combination Strategies**: Scammers frequently employ multiple techniques together to maximise profits and evade detection.

2.2.3 Prominent Rug Pull and Crypto Scam Tokens

An estimated $27 billion has been lost to cryptocurrency and NFT rug pulls and scams to date [5]. To contextualise this figure, it is comparable to the scale of some of the largest digital assets. For example, the stablecoin Tether (USDT) has a market capitalisation of approximately $167.6 billion as of August 2025 [21]. Rug-pull losses therefore represent over 16% of the value of the world's largest stablecoin. Although small relative to the entire cryptocurrency market, this magnitude demonstrates that rug-pull scams pose a systemic threat within the token economy.

The following cases represent a selection of the largest financial scams and culturally notable incidents, drawn from the Comparitech database of cryptocurrency and NFT rug pulls and scams [5]:

- **OneCoin** – $4 billion stolen: Launched in 2014, OneCoin attracted over 3 million members worldwide, with total funds estimated between $4 *billion and* 19 billion and often described as the most "successful" crypto scam.
- **Africrypt** – $3.6 billion stolen: In April 2021, Africrypt claimed to have a major attack resulting in the disappearance of $3.6 billion in Bitcoin.
- **GainBitcoin** – $3 billion stolen: India's largest crypto scam, which operated until March 2018. It promised to investors 10% monthly returns, but was unable to follow through.
- **PopCat** – $375,236 stolen: A meme-inspired rug pull in 2024 that exploited viral internet culture and online communities to lure investors.
- **TRUMP (MAGA)** – $957,552 stolen: A politically themed rug pull in January 2025. The scam illustrates how opportunistic actors exploit popular figures and movements to create tokens with temporary hype before withdrawing funds.

2.3 Machine Learning and Deep Learning in Blockchain Fraud Detection

2.3.1 Traditional Machine Learning Approaches

Classification Models

Traditional machine learning models have been utilised to handle tabular features derived from transaction statistics, opcode n-grams, and Term Frequency-Inverse Document Frequency of source code, which can capture non-linear relationships adopted in blockchain fraud detection.

- **Logistic Regression** – a supervised machine learning algorithm, a type of linear model that examines the relationship between predictor variables, which is highly relevant in performing statistical testing in the context of behavioural and social science research [22].
- **Random Forest** – a commonly-used machine learning algorithm combining the output of multiple decision trees by voting for classification or averaging to reach a final class or label [23].
- **Bagging Classifier** – a machine learning algorithm to combine the results from training multiple models independently on random subsets of the data by voting or averaging [24]
- **XGBoost (eXtreme Gradient Boosting)** – a supervised learning boosting algorithm that uses gradient boosted decision trees. It is known for its speed, efficiency and ability to scale well with large datasets. Using a level-wise tree growth strategy ensures better control over overfitting since each level balances the tree [25, 26].
- **LightGBM** – another supervised learning boosting algorithm that uses gradient boosted decision trees using a leaf-wise growth approach for focusing on a growing dataset. [26]
- **MultiOutputClassifier** – Extends classifiers to multi-label problems, allowing simultaneous prediction of different fraud categories (e.g., mint, leak, limit) [27].

Feature Engineering and Preprocessing

For raw data in blockchain fraud detection, transformation is often required, such as converting bytecode to n-grams, source code to term frequency-inverse document frequency (TF-IDF) and normalising statistical data.

- **CountVectorizer** – produces a sparse representation of the counts using to convert opcode sequences to frequency vectors (ngrams) [28].
- **TfidfVectorizer** – Captures term importance by weighting rare but informative terms of the source code. [29]
- **SimpleImputer** – Handles missing values in transaction data by replacing them with statistical estimates (e.g., mean or median). [30]
- **StandardScaler** – Normalises features to zero mean and unit variance [31]

2.3.2 Deep Learning Approaches

Recurrent Neural Networks

A recurrent neural network (RNN) is a deep neural network used for time series data that can draw conclusions based on sequential inputs [32].

- **Gated Recurrent Unit (GRU)** – A variant of RNNs that captures long-term dependencies in transaction timelines and contract execution traces, with fewer parameters than LSTMs, making it efficient for large datasets [32].

2.3.3 Late Fusion Strategies for Multimodal Models

To achieve the best accuracy and combine the prediction results from multiple individually trained models (e.g., transaction-based classifiers and bytecode-based classifiers, timeline RNN) at the decision level, a weighted average ensemble is a practical late fusion technique.

How to Implement Weighted Late Fusion Ensemble: Summary of Steps [33]

1. **Train individual models separately** on different modalities or feature sets.

2. **Evaluate each model's performance** on a validation set to determine weights. Standard metrics include accuracy, AUC, or error rate.

3. **Assign weights** proportional to these performance metrics. Better models get higher weights.

4. **Combine model predictions** on test data by computing a weighted average:

$$\hat{y} = \sum_{i=1}^{M} w_i \cdot \hat{y}_i, \qquad \text{where} \quad \sum_{i=1}^{M} w_i = 1 \tag{1}$$

5. **Make final prediction** based on the combined score (e.g., thresholding for classification)

2.3.4 Semi-Supervised Iterative Self-Learning

Semi-supervised iterative self-learning enhances model performance by leveraging large quantities of unlabeled blockchain data.

Using a semi-supervised learning technique, the process employs thresholding strategies and an ensemble decision support system. This enables incremental, unsupervised improvements to both the model and the dataset, increasing the labelled dataset size and improving classification accuracy over time. [2]

The cycle consists of:

1. **Initial Model Training** on a cleanly labelled dataset.

2. **Prediction on Unlabeled Data** using the trained model.

3. **Confidence Filtering**, where only predictions above a threshold are accepted.

4. **Dataset Expansion**, appending pseudo-labelled data to the training set.

5. **Retraining**, iteratively repeating the cycle.

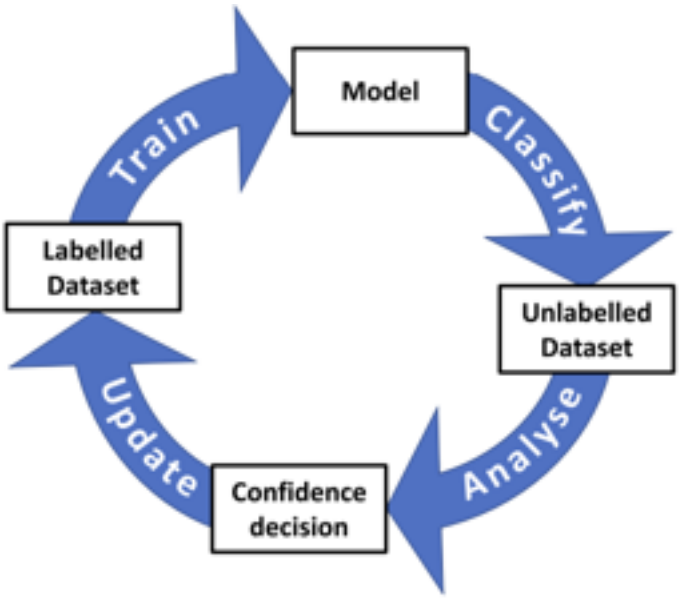

Figure 1: Iterative Learning [2]

2.3.5 Hyperparameter Optimisation with Optuna

Hyperparameter optimisation is a standard part of the machine learning process. A naive approach to hyper-parameter search is grid search, and another traditional approach is randomised search. But grid search does not return the best set and cannot be applied to deep learning. Optuna is an efficient hyperparameter optimisation framework that uses the tree-structured Parzen estimator approach by default. The Tree-structured Parzen selects the set of hyperparameters based on the history of the experiment to set the next trial. Moreover, it is easy to adapt to many use cases, such as selecting and optimising the number of layers and the number of hidden units at each layer. [34]

2.3.6 Anomaly Detection Approaches

Anomaly detection, or outlier detection, is used to identify what is normal or different from the rest of the dataset. It is also used for fraud detection, cybersecurity, quality control, and IT system management. Some of the most common ML anomaly detection algorithms include [35]:

- **Isolation Forest**: a type of decision tree that serves as an ensemble learning

method. The forest algorithm is used to detect anomalies instead of a target variable. The main idea is that more anomalous observations will have shorter paths from the root node of the tree in the forest to the leaf node. [35] [36]

- **Autoencoders**: LSTM-Autoencoder-based, a type of neural network that uses time-stamped data to identify abnormalities and detect anomalies in multivariate time-series data, generates domain constraints, and reports subsequences that violate the constraints as anomalies. [35] [37]

2.4 Existing Detection Approaches

2.4.1 Smart Contract Code Analysis

CRPWarner

Contract-related Rug Pull Warner (CRPWarner) is a detection tool designed to identify potentially malicious logic embedded in smart contracts, particularly those governing tokens. It was developed by Lin et al. [20]. Rather than observing transaction patterns or historical behaviours, CRPWarner adopts a **rule-based static analysis** approach. Smart contracts are first decompiled into an intermediate representation using the Gigahorse decompiler [38]. Over this representation, the tool applies a set of declarative rules written in Datalog, which capture code patterns associated with rug pull behaviours (e.g., hidden mint, sell restrictions, unauthorised balance changes). By relying on logic rules rather than statistical inference, CRPWarner systematically matches functional constructs in the code against a predefined taxonomy of malicious patterns.

The malicious functions that CRPWarner flags as rug pull events include:

- **Hidden Mint Function**: code paths that allow arbitrary creation of new tokens.
- **Limiting Sell Order**: restrictions that selectively prevent investors from selling tokens.
- **Leaking Token**: logic that transfers or reduces balances without the holder's consent.

In practice, its purpose is twofold: (i) to serve as a quality assurance aid for contract developers, surfacing anti-patterns before deployment, and (ii) to provide early-warning signals for auditors, regulators, or investors who wish to assess the riskiness of a token project.

CRPWarner has been evaluated on multiple datasets. On 69 open-source smart contracts related to known rug pull events [39], it achieved 91.8% precision, 85.9% recall, and an F1-score of 88.7%. In large-scale experiments on 13,484 real-world token contracts [40], written in Solidity, it maintained a precision of 84.9%.

TrapdoorAnalyser

TrapdoorAnalyser is a machine learning–based detection system trained on a dataset of 30,000 trapdoor and non-trapdoor tokens collected from UniswapV2 [41], developed by Huynh et al. [42]. The central vulnerability it targets is the *trapdoor bug*, in which a token's smart contract allows investors to buy tokens but prevents them from selling by embedding hidden restrictions or owner-only controls.

The study identifies four principal trapdoor patterns:

- **Exchange Permission (EP)**: Contracts require specific conditions to be met before sales are allowed, often selectively applied to disadvantage ordinary users.
- **Exchange Suspension (ES)**: Trading activity can be halted entirely via a Boolean flag, enabling creators to disable transfers while retaining their own privileges.
- **Amount Limit (AL)**: Contracts enforce per-transaction transfer limits (e.g., maximum token amounts), which can often be modified dynamically through backdoor functions.
- **Fee Manipulation (FM)**: Contracts impose unusually high or adjustable transfer fees, draining investor value over repeated transactions.

For classification, the authors benchmarked a range of supervised machine learning algorithms, including k-Nearest Neighbours (KNN), Support Vector Machines (SVM), Random Forests, XGBoost (XGB), and LightGBM (LGBM). Opcode sequences were first extracted from bytecode using EVDSM, and features were constructed for learning.

The dataset was randomly shuffled and split into an 80% training set and a 20% test set. Models were trained and validated using 10-fold cross-validation, with hyperparameter optimisation performed to identify the most effective algorithm.

Through this comparative process, LightGBM (LGBM) emerged as the optimal classifier across all trapdoor categories, achieving accuracy scores of up to 98%. This demonstrates that ensemble gradient boosting models are particularly well-suited for detecting subtle opcode-level patterns indicative of rug pull trapdoors.

2.4.2 Code-and-Transaction Fusion Analysis

RPHunter

RPHunter is a novel detection framework that integrates both contract code and transaction behaviour to identify Rug Pull scams, proposed by Wu et al. [3]. Unlike prior methods that focus solely on static code analysis or transaction features, RPHunter jointly models both perspectives to capture the complex interplay of malicious contract logic and suspicious trading behaviour. It is specifically designed for EVM-compatible blockchains.

RPHunter constructs two complementary graph representations:

- A **Semantic Risk Code Graph (SRCG)**, which is derived from Intermediate Representation (IR) bytecode produced by the Gigahorse decompiler [38]. The control flow and data dependencies between contract blocks are encoded, and declarative rules are used to extract potential risk indicators such as restricted transfer logic or hidden mint functions. Nodes are classified as *critical*, *invocation*, or *normal*, while edges are labelled as *critical*, *dependent*, or *normal*, reflecting their risk and relational properties.

- A **Token Flow Behaviour Graph (TFBG)**, which captures dynamic transaction activity. For each token, the earliest 500 transactions are collected to model trading behaviour before a potential Rug Pull occurs. Senders and receivers form graph nodes, with edges representing token transfers enriched by structural and behavioural features, including degree/betweenness centrality, clustering coefficients, fund in/out ratios, and short-term spikes in incoming/outgoing transfers. A Unified Aggregation GNN (UAGNN) is then applied to incorporate both node and edge

features, using a temporal masking mechanism to capture sequential dependencies in trading behaviour.

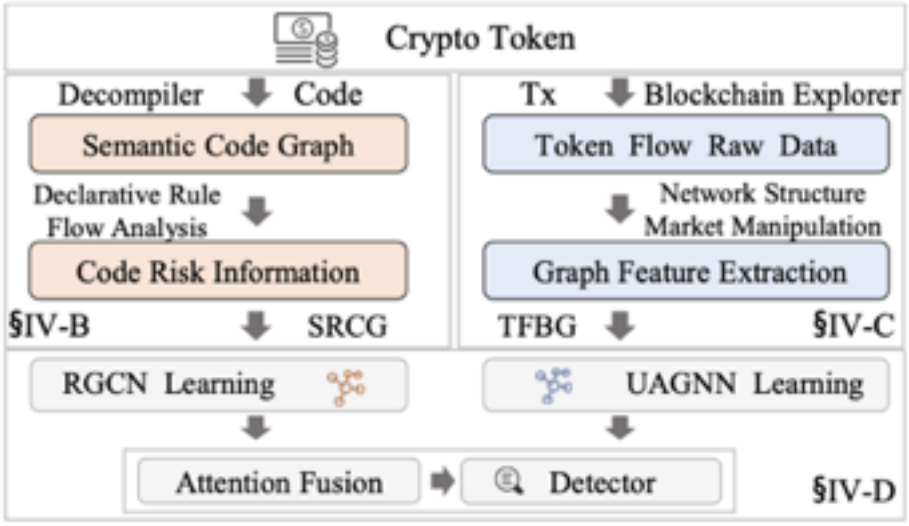

Figure 2: Overview of RPHunter architecture. [3]

These two graph embeddings are combined using an attention-based fusion model, enabling RPHunter to leverage complementary features from both code and transaction domains. For evaluation on their ground-truth dataset of 645 manually analysed Rug Pull incidents and 1,675 benign tokens, RPHunter achieved a precision of 95.3%, recall of 93.8%, and F1-score of 94.5% [3], substantially outperforming state-of-the-art rule-based and statistical transaction methods. In real-world deployment on Ethereum Mainnet, the system identified 4,801 Rug Pull tokens with an overall precision of 90.7%.

The authors categorise Rug Pull code risks into three groups:

- **Sale Restrict:** mechanisms that selectively prevent sales, such as *Amount Restrict (AR)*, *Timestamp Restrict (TR)*, and *Address Restrict (ADDR)*. Although AR may appear to be a legitimate "anti-whale" feature, when combined with hidden TR or ADDR logic, it can deceptively signal investor safety while secretly enabling liquidity withdrawal by developers.

- **Variable Manipulation:** owner-controlled modifications of key variables, such as *Modifiable Tax Rate (MTR)*, *Modifiable Tax Address (MTA)*, and *Modifiable External Call (MEC)*, which can redirect funds or contract logic to malicious destinations.

- **Balance Tamper:** concealed functions such as *Hidden Mint/Burn (HM)* or *Hidden Balance Modification (HBM)* that enable arbitrary creation, destruction, or reassignment of tokens.

A key insight from Wu et al.'s analysis is that Rug Pull scams often combine "good faith" features (e.g., AR disclosed as a safety mechanism) with undisclosed backdoors (e.g., TR or HBM) to build false investor confidence. By detecting these combinations through both code and early transaction analysis, RPHunter provides robust early warnings of scam tokens before catastrophic losses occur.

2.4.3 Comparative Benchmarks and Limitations

Tool	Performance	Strengths	Limitations
CRPWarner [20]	69 contracts: Precision: 91.8%, Recall: 85.9%, F1: 88.7%. 13,484 real-world: Precision: 84.9%	- Logic-based detection via Datalog. - Identifies hidden mint, sell restriction, and leaking tokens.	- Requires bytecode/source. - Limited coverage (only three malicious patterns). - Not scalable to unseen scams.
TrapdoorAnalyser [42]	30,000 tokens on UniswapV2. LightGBM accuracy $\approx$ 98%	- ML-driven classification using opcode features. - Detects trapdoor families (EP, ES, AL, FM).	- Narrow focus (trapdoors only). - Relies heavily on opcode features. - Lacks transaction behaviour analysis.
RPHunter [3]	Precision: 95.3%, Recall: 93.8%, F1: 94.5%	- Fuses code & transaction features (SRCG + TFBG). - Captures static risks & dynamic behaviours.	- High computational overhead. - Needs source code, which is only 1% of the public [43]

Table 1: Comparative benchmarks and limitations of rug pull detection approaches

From the comparison table, CRPWarner is precise but limited in coverage, TrapdoorAnalyser achieves high accuracy but is narrow in focus. At the same time, RPHunter offers the most comprehensive detection at the cost of computational complexity.

Chapter 3

Methodology

This chapter outlines the methodology adopted for designing and implementing a scalable rug pull detection framework. It describes the techniques selected, their justification based on insights from the literature review, and the rationale for how they address the gaps identified in existing approaches. The focus is on detailing the datasets, model design, self-learning strategy, and system architecture that together form the foundation of the framework.

3.1 Identified Gaps

The preceding literature review outlined automated rug pull detection approaches, from logic-based tools like CRPWarner, to machine learning classifiers like TrapdoorAnalyser, and multimodal frameworks like RPHunter. While these studies show that automated detection is feasible, they reveal significant shortcomings in data availability, feature integration, adaptability, and scalability.

Despite advancements in rug pull detection, critical challenges hinder the practical and real-world deployment of these techniques. These limitations present opportunities to develop and demonstrate a scalable detection framework within this project:

- **Limited labelled datasets**: Publicly available ground truth datasets are scarce, fragmented, and often focused on narrow scam patterns (e.g., ground truth from CRPWarner, RPHunter, and Trapdoors). This restricts the generalisation of models to unseen types of rug pulls. While the creation of new large-scale datasets is

beyond the scope of a single dissertation, it is feasible to consolidate and refine existing labelled datasets to improve their usability.

- **Low availability of verified source code**: Only about 1% of the smart contracts on Ethereum are verified and open-source [43], meaning that most approaches that rely heavily on source code or annotated bytecode cannot scale to real-world settings. A practical approach is to focus on bytecode and transaction-level data, which are more widely available.

- **Cross-chain generalisation**: Existing models are mostly evaluated on Ethereum ERC-20 tokens. However, rug pulls frequently occur on other blockchains such as BNB Chain, Polygon, or Solana. There is little evidence that current detection methods generalise across ecosystems [44]. Full multi-chain validation may be out of reach within this project, but comparative testing across a small number of representative chains is feasible.

- **Feature integration challenges**: Code-level, transaction-level, and graph-based features are usually analysed separately. Few frameworks provide a unified fusion approach that can effectively combine static and dynamic behavioural signals. Exploring integration at a limited scale, particularly through ensemble or fusion models, is achievable within this research scope.

- **Adaptability to evolving scams**: Attackers frequently introduce new scam mechanisms, making static detection pipelines obsolete. Current approaches lack continuous learning capabilities to incorporate new data and adapt dynamically. While it is not possible to anticipate every novel scam, combining mechanisms for incremental updates or pseudo-labelling is a feasible step towards adaptability.

- **High computational cost**: Graph-based and fusion models (e.g., RPHunter) achieve high accuracy but are expensive to run at scale, raising concerns for real-time deployment in production systems. Although large-scale optimisation is beyond scope, efficiency trade-offs and lightweight alternatives can be explored.

Collectively, these limitations emphasise the need for a scalable, multi-source, and

self-learning detection framework that can incrementally improve with new data, operate across chains, and integrate diverse features for robust rug pull identification.

This leads directly to the main research question of this dissertation:

> *Can rug pull scams be automatically detected and categorised using blockchain transaction data—even when source code or bytecode is unavailable—through a scalable, self-learning machine learning system that adapts to emerging scams and supports cross-chain deployment?*

3.2 Development Objectives

In order to address the identified gaps and answer the central research question, this dissertation pursues the following objectives:

1. **Design a scalable detection framework** that integrates multiple data sources, including code-level, transaction-level, graph-based, and sequential features.
2. **Develop semi-supervised self-learning mechanisms** (e.g., pseudo-labelling) that can incrementally expand training datasets with high-confidence predictions, improving detection accuracy over time.
3. **Support incremental learning and adaptability**, enabling the system to incorporate new rug pull strategies and extend classification to newly emerging scam categories.
4. **Enable cross-chain generalisation**, validating the system across multiple blockchain ecosystems such as Ethereum, BNB Chain, and Polygon.
5. **Evaluate and test performance** on benchmark datasets, with comparisons to reported results from tools such as CRPWarner, TrapdoorAnalyser, and RPHunter, using metrics including precision, recall, F1-score, runtime efficiency, and scalability.

Following the research objectives, this project makes use of datasets from three significant research efforts. Each dataset contributes a different perspective on rug pull behaviour, enabling incremental scaling from small-scale proof-of-concept evaluation to broader multi-class testing.

3.3 Dataset

The reliability of any machine learning framework depends heavily on the quality and validity of its datasets. For this project, datasets were sourced from peer-reviewed research and publicly available repositories that are widely cited in the rug pull detection literature. To ensure reproducibility and methodological alignment, all datasets were obtained directly from their original publications or official repositories, and only those with consistent labelling schemes were included. In this way, the data used in this study is both trustworthy and suitable for feature extraction, model training, and evaluation.

Three primary datasets were selected to support the development and evaluation of the proposed framework, each contributing a complementary perspective on rug pull behaviours, as follows:

- **CRPWarner Dataset** – Provided by Lin et al. [20], this dataset includes 69 manually labelled smart contracts (ground truth) [45] covering three rug pull behaviours: Mint, Leak, and Limit. In addition, a partially labelled subset from the CRPWarner large dataset is used, consisting of around 90 contracts per class [46]. This dataset forms the foundation for initial supervised training and benchmarking.

- **Trapdoor Dataset** – Introduced by Huynh et al. [42], this dataset contains approximately 30,000 labelled contracts deployed on UniswapV2, annotated into five categories: Amount Limit (AL), Exchange Permission (EP), Exchange Suspension (ES), Fee Manipulation (FM), and Indirect Control (IC) [47]. This dataset enables large-scale opcode feature analysis and evaluation of model generalisation across different scam strategies.

- **RPHunter Dataset** – From Wu et al. [3], this dataset contains approximately 700 labelled contracts divided into eight categories: Amount Restrict (AR), Timestamp Restrict (TR), Address Restrict (ADDR), Modifiable Tax Rate (MTR), Modifiable Tax Address (MTA), Modifiable External Call (MEC), Hidden Mint/Burn (HM), and Hidden Balance Modification (HBM) [48].

This project initially began with the CRPWarner ground truth dataset and subsequently expanded to the CRPWarner sample, RPHunter, and Trapdoor datasets in

sequence. Additional raw data was collected via the Etherscan V2 API, which provides verified source code (when available), compiled bytecode, and historical transaction logs for each contract [6, 49].

3.4 Model Design

The model design in this project consists of two complementary components: a supervised classifier that categorises rug pull scams into known labels, and an anomaly detector that flags suspicious behaviours outside of the labelled taxonomy. This dual design addresses both the need for accurate classification and the challenge of detecting novel or evolving scams, as shown in Figure 3.

3.4.1 Classifier Models

The classifier is responsible for learning from labelled datasets (CRPWarner, Trapdoor, and RPHunter) and predicting rug pull categories. A range of models is explored to capture different feature modalities:

- **Tabular features** (statistical data) processed via imputation, scaling, and Optuna-tuned selected static ML models such as Logistic Regression, Random Forest, XGBoost, and LightGBM.
- **Source code features** (TF-IDF, n-grams) fed into the same family of static ML models.
- **Bytecode features** (opcode frequency, entropy, n-grams) transformed into float vectors and processed with ML classifiers.
- **Sequential features** (transaction time series) modelled using GRU-based RNNs.

Since no single modality fully represents rug pull behaviour, a late-fusion ensemble (**Fusion Model**) is designed. In this approach, the outputs of individual models—such as those trained on bytecode features, transaction statistics, or timeline sequences—are combined into a single prediction. Each model contributes proportionally according to

a weight that reflects its standalone performance, with stronger models exerting greater influence on the ensemble decision. These weights are not assigned manually but are optimised automatically using Optuna during validation, ensuring that the ensemble adapts dynamically to the relative strengths of its components.

Formally, the ensemble prediction $\hat{y}$ is defined as:

$$\hat{y} = \sum_{i=1}^{M} W_i \cdot \hat{y}_i \quad \text{where} \quad \sum_{i=1}^{M} W_i = 1$$

Here, $\hat{y}_i$ represents the prediction of the i-th model, W_i is its corresponding weight, and M is the total number of models in the ensemble. The formal optimisation of these weights, together with illustrative examples of their configurations, is presented in Chapter 4.

3.4.2 Model Selection

The tabular, opcode, and TF–IDF feature spaces in this project are heterogeneous (dense/sparse; nonlinear; small-to-mid scale). The set of classifiers should have distinct inductive biases and robustness characteristics. All models output calibrated probabilities to support late-fusion weighting.

Criteria

To decide which models are promoted into the fusion set, we apply a multi-criteria filter on a held-out validation split (stratified) with Optuna hyperparameter optimisation. The primary criterion is *macro-F1* (averaged across labels), reflecting balanced performance under class imbalance. We use the following tie-breakers, in order:

1. **Calibration quality**: Brier score and reliability diagrams (lower is better).
2. **Efficiency**: training time per trial and inference latency per sample (reported in Chapter 4).
3. **Robustness**: stability across 5-fold CV (mean ± std of macro-F1) and tolerance to missing features (via imputation).

4. **Compactness & interpretability**: model size and feature attribution availability (e.g., LR coefficients, tree importances).

Only candidates that pass a minimum calibration threshold and remain within a pragmatic latency budget will be promoted to the fusion pool and produced *calibrated probabilities* (Platt scaling or isotonic regression) to enable meaningful late-fusion weighting.

Model Candidate

- **Logistic Regression (LR)** (`sklearn.linear_model.LogisticRegression`) [50]
  A strong linear baseline for both sparse (TF–IDF, opcode n-grams) and dense tabular features. It can be well-calibrated with simple regularisation (L1/L2), and its coefficients aid interpretability.

- **Random Forest (RF)** (`sklearn.ensemble.RandomForestClassifier`) [51]
  It's robust to feature scaling and outliers, and can handle nonlinear interactions every day in behaviour features (e.g., statistics from transactions). Good variance control through the number/depth of trees.

- **Bagging Classifier** (`sklearn.ensemble.BaggingClassifier`) [52]
  Variance-reduction via bootstrapping around a strong or weak base estimator (e.g., shallow trees). Useful on small/medium datasets (e.g., CRPWarner Groundtruth) to stabilise decision boundaries without heavy hyperparameter complexity.

- **MLP (Feed-forward NN)** (`sklearn.neural_network.MLPClassifier`) [53]
  A compact feed-forward neural model for non-linear decision surfaces on dense, engineered features (e.g., byte/opcode frequency, aggregated transaction stats). It can exploit interactions that linear models miss while remaining lightweight compared to Deep learning models.

- **XGBoost (XGB)** (`xgboost.XGBClassifier`) [54]
  Gradient-boosted trees excel on tabular data and tolerate missingness. It can capture complex feature interactions with strong generalisation on medium-sized

datasets, often outperforming bagging in accuracy at similar inference cost. Useful class-imbalance controls.

- **LightGBM (LGBM)** (`lightgbm.LGBMClassifier`) [55]
  Histogram-based, leaf-wise boosting with excellent speed–accuracy trade-off and native handling of large, sparse feature spaces (e.g., TF–IDF/opcode n-grams).

Optuna-based model selection

1. **Split & metrics**: stratified train/validation; objective = validation macro-F1 (multi-label via per-label averaging).
2. **Define candidates**: LR, RF, Bagging, XGB, LGBM, MLP (classical); GRU for sequences; TF–IDF/CountVectorizer for text/opcode.
3. **Trial loop**: sample hyperparameters from predefined spaces; build the modality-specific pipeline; fit on train; evaluate on validation (macro-F1).
4. **Pruning**: stop underperforming trials early.
5. **Refit & calibrate**: refit best params on train+val; apply probability calibration on a clean inner split.
6. **Fusion tuning**: optimise non-negative weights W_i with $\sum_i W_i = 1$ on validation; report final metrics and latency.
7. **Budget**: fixed n_{trials} with pruning.

Key tuned parameters

- **Vectorizers**
  - `ngram_range` (opcode): choose $(1,1)$, $(1,2)$, or $(1,3)$; captures local opcode patterns.
  - `max_features` (both): vocabulary cap (10–20,000); trades coverage vs. sparsity.
  - `min_df` (both): minimum document frequency (1–5); filters rare/noisy tokens.

- **Logistic Regression (LR)**
  - `penalty` $\in \{\ell_2, \ell_1, \text{elasticnet}\}$: regularisation type.
  - `C` (10^{-4}–10; log scale): inverse regularisation strength.
  - `l1_ratio` (0–1): ℓ_1 mixing (elastic net only).
  - `tol` (10^{-10}–10^{-2}): optimisation tolerance.
- **Random Forest (RF)**
  - `n_estimators` (50–1000): number of trees.
  - `max_depth` (5–30): tree depth (controls bias–variance).
  - `criterion` $\in$ {gini, entropy, log_loss}: split quality measure.
- **Bagging**
  - `n_estimators` (50–1000): number of bootstrap models (variance reduction).
- **XGBoost (XGB)**
  - `n_estimators` (50–1000): boosting rounds.
  - `max_depth` (5–30): tree depth per round.
  - `learning_rate` (0.01–1.0): step size (lower = steadier, slower).
  - `subsample` (0.5–1.0): row sampling per round (regularisation).
  - `colsample_bytree` (0.5–1.0): feature sampling per tree.
- **LightGBM (LGBM)**
  - `n_estimators` (50–1000), `max_depth` (5–30), `learning_rate` (0.01–1.0): as above.
  - `subsample` (0.5–1.0): row sampling (bagging).
  - `colsample_bytree` (0.5–1.0): column sampling.
- **MLP**

  - `hidden_layer_sizes` ∈ {(32), (64), (128), (256), (32,64), (64,32), (64,64), (64,128), (128,128), (32,64,128)}: capacity.
  - `activation` ∈ {relu, tanh, identity, logistic}: nonlinearity.
  - `learning_rate_init` (10^{-10}–10^{-1}): initial step size.

- **GRU (sequence classifier; via `TSBlocks`)**
  - `units`: GRU width (model capacity).
  - `learning_rate`: Adam step size.
  - `batch_size`, `epochs`: training budget (with early stopping / LR scheduling).

- **Fusion weights** (W_i)
  - Simplex-constrained non-negative weights ($\sum_i W_i = 1$): relative contribution of calibrated model outputs; tuned to maximise validation macro-F1.

3.4.3 Anomaly Detection

In addition to supervised classification, this project incorporates an anomaly detection module designed to capture contracts whose behaviour does not fit any of the labelled scam categories. This provides resilience against novel or evolving rug pull strategies that fall outside the current taxonomy.

- **Tabular Features**: Transaction statistics (e.g., counts, ratios, gas metrics) are preprocessed with imputation and scaling, then analysed using an **Isolation Forest** (`sklearn.ensemble.IsolationForest` [56]). This ensemble method isolates anomalies by partitioning the feature space; contracts requiring fewer splits to isolate are flagged as anomalous.

- **Source Code Features**: TF–IDF and n-gram vectors of verified Solidity code are also passed into an **Isolation Forest**, enabling the detection of unusual lexical or structural patterns compared with the distribution of standard contracts.

- **Bytecode Features**: Opcode frequency and n-gram vectors are converted to dense float representations and analysed by an **Isolation Forest**. This allows anomaly

detection even when source code is unavailable, leveraging patterns in compiled bytecode.

- **Sequential Features**: Transaction timelines (sequences of calls, gas usage, timestamps) are padded to a fixed length and passed into a **Bi-LSTM Autoencoder**. The model is trained to reconstruct typical transaction sequences; a high reconstruction error indicates anomalous behaviour.

Outputs from these modality-specific detectors are combined through an **anomaly fusion layer**, which applies Optuna-tuned weights and a global threshold to produce a final anomaly score or binary anomaly flag. This ensemble design ensures that contracts exhibiting abnormal behaviour in any feature space can still be flagged, even if they do not match a known rug pull pattern.

Flow

1. Train per-modality detectors: Isolation Forest (tabular, TF–IDF, opcode) and a BiLSTM autoencoder (sequences, masked padding).

2. Score: IF uses $s_{\mathrm{IF}}(x) = -\mathrm{score\_samples}(x)$; AE uses per-sequence reconstruction error $s_{\mathrm{AE}}(x) = \frac{1}{T}\sum_t \|x_t - \hat{x}_t\|_2^2$.

3. Thresholding: if weak labels exist, pick τ to maximise validation F1; otherwise choose a quantile q (e.g., 97.5^{th}).

4. Fusion: combine normalised scores via non-negative weights v_j with $\sum_j v_j = 1$, tuned on validation when labels exist (else equal weights).

5. **Output:** anomaly flag

$$\mathrm{flag} = \begin{cases} 1, & \text{if } \sum_j v_j s_j \geq \tau_{\mathrm{global}}, \\ 0, & \text{otherwise,} \end{cases}$$

plus the per-modality scores $\{s_j\}$ for interpretability.

Key tuned parameters

- **Isolation Forest** (`sklearn.ensemble.IsolationForest`)
  - `n_estimators` (trees): model capacity; higher reduces variance.
  - `max_samples` (row subsample ratio): regularises each tree.
  - `max_features` (feature subsample ratio): adds feature-level randomness.
  - `contamination` (anomaly fraction): sets default threshold if labels are absent.
  - `bootstrap` (True/False): enables bagging-style stability.
- **Sequence AE** (BiLSTM; via `TSBlocks`)
  - `units` (hidden width): capacity of encoder/decoder.
  - `learning_rate` (Adam): optimisation step size.
  - `batch_size`, `epochs`: training budget with early stopping & LR scheduling.
  - `window length` T: sequence context; interacts with masking/padding.
  - **AE threshold** τ_{AE}: reconstruction-error cutoff (F1-optimised or quantile).
- **Score fusion & thresholds**
  - **Weights** $v_j \geq 0$, $\sum_j v_j = 1$: per-modality contribution to final score.
  - **Global threshold** τ_{global}: tuned on validation (or set by quantile when unlabeled).
  - **Normalisation**: min–max or rank-based scaling of s_j to $[0, 1]$ before fusion.

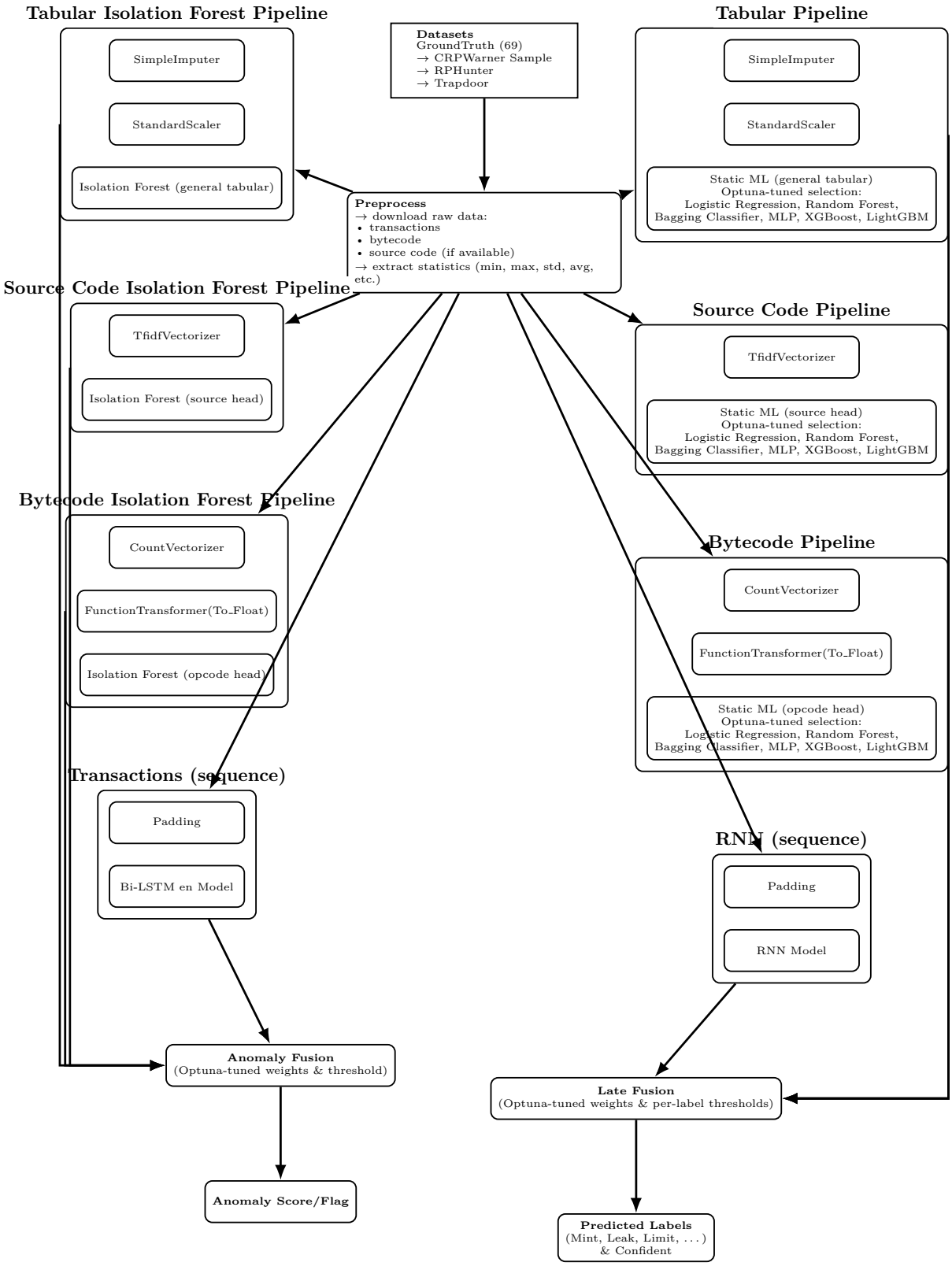

Figure 3: Training pipeline for multi-modal rug-pull detection: classify fusion and anomaly fusion.

3.5 Self-Learning: to expand the label or the dataset

As discussed in Section 3.1, the framework is constrained by limited labelled data and by evolving scam behaviours that exceed the capacity of static models to adapt. We address these constraints using a self-learning (pseudo-labelling) approach that expands the effective training set and enables gradual incorporation of new labels.

As shown in Figure 4, the loop: (1) trains on the current ground truth, (2) predicts calibrated probabilities on previously unseen contracts, (3) admits only *high-confidence* pseudo-labels using per-label high/low thresholds into a quarantine buffer, and (4) accepts the candidate dataset *only if* a preview Optuna run improves validation macro-F1 over a cached baseline by at least Δ_{accept}; otherwise it is discarded and thresholds are tightened. Existing labels are never overwritten, and uncertain rows are exported with probabilities for audit.

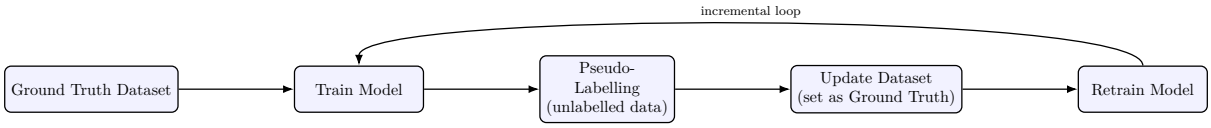

Figure 4: Self-learning loop: expanding ground truth

3.5.1 Procedure

1. **Inputs.** Load a labelled *source* CSV (ground truth) and an unlabelled *target* CSV; identify the set of existing labels (*old labels*). Normalise label columns to $\{-1, 0, 1\}$, where -1 denotes unknown.

2. **Predict.** For target addresses not present in source, infer per-label probabilities $p_c(x) \in [0, 1]$ using the current best classifier (calibrated).

3. **Thresholding (asymmetric).** For each label c with thresholds $(\texttt{low}_c, \texttt{high}_c)$:

$$\hat{y}_c(x) = \begin{cases} 1 & \text{if } p_c(x) \geq \texttt{high}_c, \\ 0 & \text{if } p_c(x) \leq \texttt{low}_c, \\ -1 & \text{otherwise (uncertain).} \end{cases}$$

Only unknown entries (-1) may be filled; existing 0/1 values are preserved.

4. **Staging.** Append only the new, fully decided rows (all old labels $\in \{0, 1\}$) to form a *candidate* ground truth; export uncertain rows (with p_c columns) and skipped addresses (with reasons).

5. **Acceptance gate (preview tuning).** Compute a cached *baseline* macro-F1 on a fixed validation split, then run a small Optuna preview on the candidate to obtain $\mathrm{F1}_{\mathrm{preview}}$. Accept the candidate iff

$$\mathrm{F1}_{\mathrm{preview}} \;\geq\; \mathrm{F1}_{\mathrm{baseline}} + \Delta_{\mathrm{accept}} \,.$$

6. **Finalise (optional).** If accepted and enabled, retrain with a heavier Optuna budget, save the updated model, and update downstream artefacts; otherwise discard the candidate.

7. **Iterate.** Repeat optionally in chunks until no further improvement or until the pool of unlabelled data is exhausted.

3.5.2 Key parameters

The pipeline is implemented by `SelfLearningService.run` (candidate construction, acceptance gate, finalisation) and `ExpandLabelsService.run` (per-label threshold fulfilment and confident-only export). Configuration fields align as follows:

- **`low, high`** (scalars or per-label maps; default 0.10/0.90): asymmetric bounds for confident 0/1 decisions; values in (`low`, `high`) remain -1.
- **`preview_trials`** (e.g., 8): fast Optuna trials to test whether the candidate improves validation macro-F1 (acceptance gate).
- **`n_trials`** (e.g., 50): heavier Optuna budget executed *only if* the candidate is accepted and `do_train` is True.
- **`accept_min_delta`** (Δ_{accept}; default 0): minimum macro-F1 gain over baseline required to accept the candidate.

- `test_size` (e.g., 0.2): validation split used for computing baseline and preview metrics.
- `chunk_size`, `chunk_index`: process large targets in bounded slices; limits the blast radius of any mislabels and respects external API quotas.
- **Artefacts:** automatically writes *pseudo* (accepted rows), *uncertain* (with p_c), *skipped* (reasons), and JSON *logs* for auditability.

3.5.3 Safeguards and error control

Self-training risks *confirmation bias* and *compounding errors*. We employ multiple guards:

1. **No overwrite policy:** Existing 0/1 labels are immutable; only unknowns (-1) may be filled.
2. **High/low thresholds:** Asymmetric per-label bounds avoid forcing decisions in ambiguous regions.
3. **Validation gate:** Candidates are accepted only if preview macro-F1 meets the improvement criterion; otherwise rolled back.
4. **Quarantine buffer:** New rows are staged and evaluated before being merged; uncertain rows are exported for manual/aided review.
5. **Chunking:** Limits per-round change; reduces propagation of potential mislabels.
6. **Calibration checks:** Only calibrated probabilities are thresholded; low-calibration models are excluded.

3.6 Architecture

As shown in Figure 5, the project integrates the FastAPI backend, a Next.js frontend, and supporting services such as Redis caching, a model registry, and a meta-service for version control. The system relies on external data sources, including the Etherscan V2 API and the 4byte signature database. Wrapping all components in Docker containers

ensures that dependencies and runtime configurations are isolated, reducing conflicts and making it straightforward to reproduce.

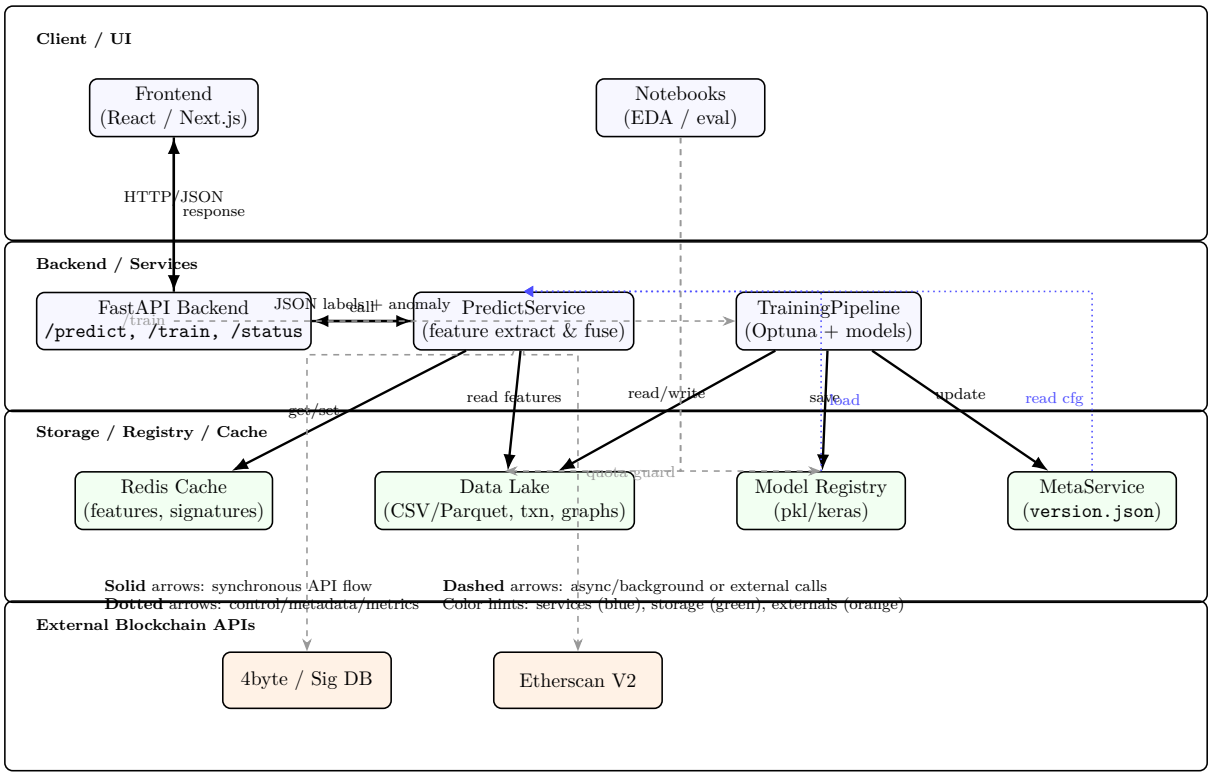

Figure 5: System architecture for the rug-pull detection platform

Components:

- **Frontend:** Next.js page, a lightweight demo interface to visualise prediction results. A snapshot is provided in Appendix A (Figure 12)
- **Backend:** FastAPI container exposing `/train`, `/predict`, and `/status` endpoints. A snapshot is provided in Appendix A (Figure 13)
- **Services:** Trainer and PredictService containers with Optuna tuning, feature extraction, and inference logic.
- **Storage:** Redis (cache), model registry, and data lake containers.
- **External Interfaces:** Calls to Etherscan V2 and 4byte for getting raw data like bytecode, Ethereum signature information, etc. inside the backend container.

The code and models are provided on both the GitLab of the university [57] and GitHub [58].

Chapter 4

Operation & Experiment

4.1 Experiment Setup

4.1.1 Computing Environment

All experiments were conducted on a personal workstation with the following specifications:

- **Processor:** 2.3 GHz Quad-Core Intel Core i7
- **Graphics:** Intel Iris Plus Graphics 1536 MB
- **Memory:** 32 GB 3733 MHz LPDDR4X
- **Operating System:** macOS Sequoia 15.6
- **Startup Disk:** Macintosh HD
- **GPU capability:** No dedicated GPU was available; training the full pipeline on the consolidated dataset took approximately 8–10 hours for 15 Optuna trials on CPU only (excluding preprocessing such as fetching bytecode/source/transactions). The codebase *can* use a GPU for the deep components (e.g., GRU/GNN timeline models) with minor changes—running on an NVIDIA CUDA machine (or CUDA-enabled cloud instance), installing GPU builds of PyTorch/TensorFlow with matching CUDA/cuDNN, and moving models/tensors to the GPU (e.g., `model.to("cuda")`,

`autocast` for mixed precision). Classical scikit-learn models (Logistic Regression, Random Forest, TF–IDF pipeline) remain CPU-bound unless replaced with GPU equivalents (e.g., RAPIDS cuML). On the current Intel-based macOS system, CUDA is not supported; therefore, GPU acceleration requires migrating to a CUDA-capable host.

Current Models

The models used in this experiment were trained on a consolidated dataset (`groundtruth_v8.csv`) comprising 1,550 records. This dataset combined verified ground truth with pseudo-labelled samples drawn from CRPWarner, RPHunter, and Trapdoor sources. Ensuring coverage across all 17 labels:

- Address Restrict
- Amount Restrict
- Hidden Balance Modification
- Hidden Mint/Burn
- Leak
- Limit
- Mint
- Modifiable External Call
- Modifiable Tax Address
- Modifiable Tax Rate
- TimeStamp Restrict
- Amount Limit
- Exchange Permission
- Exchange Suspension
- Fee Manipulation
- Invalid Callback

Trapdoor

- **Classifier models (selected via Optuna):** The choice of model for each feature type was made automatically through Optuna hyperparameter optimisation, which evaluated a pool of candidate classifiers and selected the best-performing option for each modality. The resulting selections also align with the nature of the features:
  - **Statistic features: Logistic Regression** — robust on low-to-medium dimensional tabular features, provides calibrated probabilities and interpretability.
  - **TF–IDF (Solidity source code): Logistic Regression** — linear models with regularisation are well-suited to sparse, high-dimensional text data, and Logistic Regression consistently outperformed alternatives.

  - **3-grams (opcode sequences): Random Forest** — handles sparse n-gram counts and feature interactions effectively without requiring heavy tuning, making it suitable for opcode sequence features.
  - **Timeline features: GRU / GNN sequence models** — recurrent and graph-based sequence models capture temporal dependencies and evolving token behaviours. The hidden layer size and other hyperparameters were tuned by Optuna.

- **Fusion ensemble (Optuna-optimised weights):** Since no single modality captures all rug pull behaviours, Optuna was also used to optimise late-fusion weights, ensuring that each model contributes proportionally to its predictive reliability:
  - Statistic features: 0.787
  - TF-IDF (Solidity): 0.242
  - Opcode 3-grams: 0.259
  - Timeline: 0.466

- **Anomaly detection fusion (Optuna-optimised weights):** For binary anomaly detection, a separate optimisation was performed. Here, recall was prioritised to minimise the chance of missing anomalous behaviours, even at the expense of precision:
  - Statistic features: 0.766
  - TF-IDF (Solidity): 0.195
  - Opcode 3-grams: 0.566
  - Timeline: 0.130

4.2 Evaluation Metrics

Performance was measured using:

- **Precision, Recall, F1-score** (macro, micro averaged, and accuracy for anomaly).
- **Confusion matrices** for per-class breakdowns.

4.3 Baseline Results

All evaluation steps are available as a Jupyter notebook (`notebooks/4_eval.ipynb`) in the project repository [57, 58], enabling full reproducibility of the tables and figures reported in this chapter.

4.3.1 CRPWarner Ground Truth

Classification Results

The framework was first evaluated on the CRPWarner ground truth dataset containing labels for Mint, Leak, and Limit behaviours. Performance metrics are summarised in Table 2, while Figure 6 presents the corresponding confusion matrices.

Class	Precision	Recall	F1-score	Support
Mint	0.80	1.00	0.89	20
Leak	0.89	0.89	0.89	9
Limit	0.97	0.97	0.97	30
micro avg	0.89	0.97	0.93	59
macro avg	0.89	0.95	0.91	59

Table 2: Classification report on CRPWarner ground truth dataset.

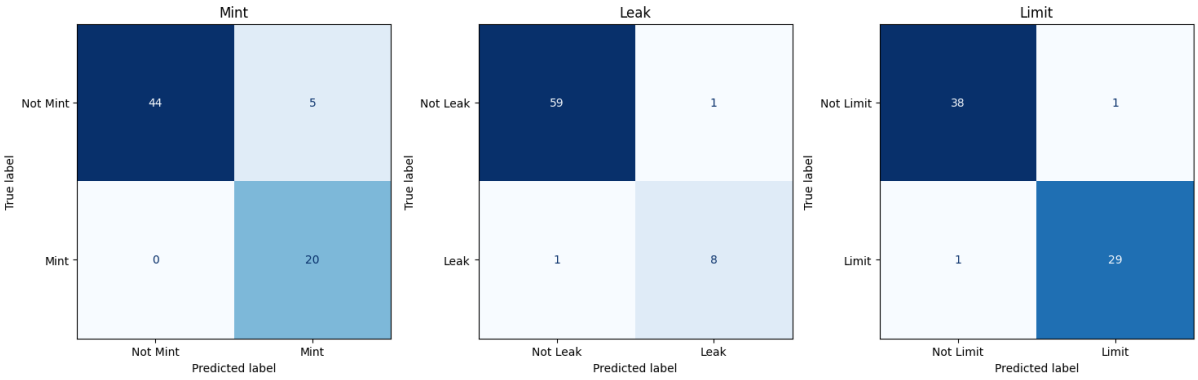

Figure 6: Confusion matrices for Mint, Leak, and Limit predictions on CRPWarner ground truth dataset.

The results demonstrate strong predictive performance across all three categories. *Limit* achieved the highest overall precision and recall (F1 = 0.97), while *Mint* achieved perfect recall (1.00) but lower precision (0.80), indicating a tendency to over-predict Mint cases. *Leak* achieved balanced precision and recall (both 0.89). Therefore, it should

be supplemented with more verified Mint cases (even via pseudo-labelling with stricter confidence thresholds).

Anomaly Detection

Anomaly detection collapsed labels into binary (*normal* vs *anomaly*). Results are shown in Table 3.

Class	Precision	Recall	F1-score	Support
normal	0.00	0.00	0.00	27
anomaly	0.61	1.00	0.76	42
accuracy			0.61	69

Table 3: Anomaly detection results on CRPWarner ground truth dataset.

The anomaly fusion successfully identified all anomalous contracts (**recall = 1.00**) but also produced a high false positive rate (**precision = 0.61**), resulting in no correctly classified normal cases. This demonstrates that anomaly detection in isolation is not reliable for precise classification. Its main strength, however, lies in perfect recall: no anomalous contracts escape detection. Such high sensitivity is valuable in financial security, where the cost of overlooking a malicious contract far outweighs the inconvenience of false positives. The drawback is a lack of specificity, as the system flagged all normal contracts as anomalous, reflecting a broader limitation of anomaly detection methods, which are inherently biased towards over-flagging when representative training data is limited.

Within the broader framework, anomaly detection therefore acts as a *fail-safe mechanism*—flagging suspicious contracts outside the labelled taxonomy and ensuring that no anomalous behaviours are missed. This design is particularly useful for emerging rug pull strategies that are not yet represented in the training data. In practice, the component is best deployed as a triage mechanism: contracts flagged as anomalous should be routed for human review rather than treated as confirmed scams. In this way, anomaly detection complements the more precise but taxonomy-limited classifiers, helping surface novel attack strategies for further investigation.

4.3.2 CRPWarner Large Sample: Mint

Classification Results

The model was further evaluated on the CRPWarner large sample containing Mint-labelled contracts. Performance metrics are summarised in Table 4, while Figure 7 shows the corresponding confusion matrix.

Class	Precision	Recall	F1-score	Support
No Mint	1.00	0.43	0.60	14
Mint	0.91	1.00	0.95	78
accuracy			0.91	93

Table 4: Classification report on CRPWarner large sample (Mint).

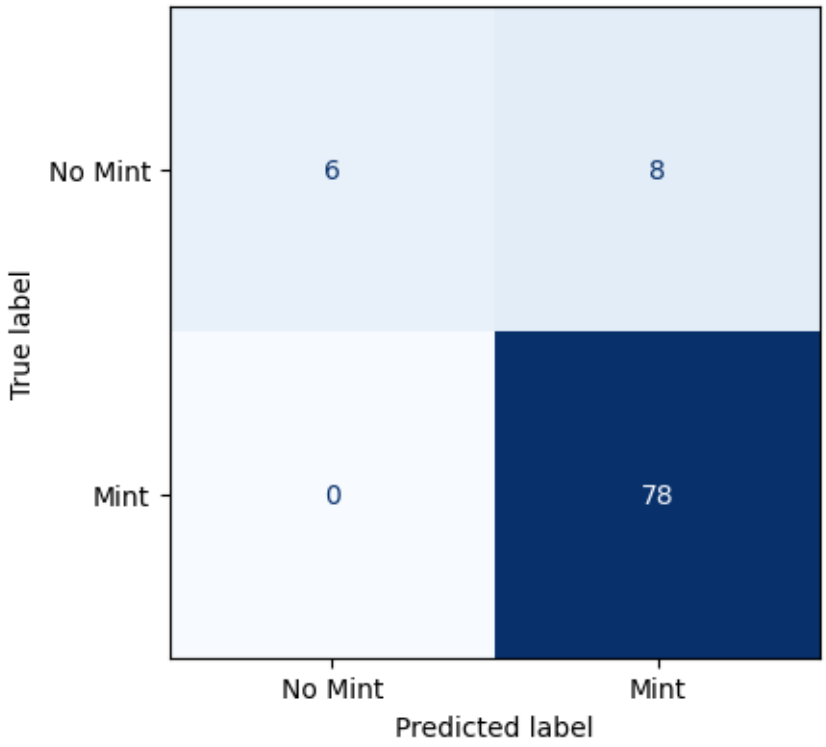

Figure 7: Confusion matrices for Mint predictions on CRPWarner large sample: mint dataset.

The classifier achieved near-perfect recall for Mint contracts (1.00), ensuring that no true Mint cases were missed. However, several non-Mint contracts were misclassified as Mint, lowering the precision to 0.91 and the recall for the No Mint class to 0.43. This indicates that while the model is highly sensitive to Mint behaviours, it also tends to

over-predict them. One reason for this is the relatively low decision threshold used during validation for the Mint class ($\tau = 0.342$). This threshold favoured recall—ensuring that Mint contracts were not overlooked—but at the expense of precision, as more borderline cases were classified as Mint. Adjusting the threshold upward would improve the precision of Mint detection, though with some trade-off in recall.

4.3.3 CRPWarner Large Sample: Leak

Classification Results

The model was further evaluated on the CRPWarner large sample containing Leak-labelled contracts. Performance metrics are summarised in Table 5, while Figure 8 shows the corresponding confusion matrix.

Class	Precision	Recall	F1-score	Support
No Leak	0.93	1.00	0.97	14
Leak	1.00	0.99	0.99	74
accuracy			0.99	88

Table 5: Classification report on CRPWarner large sample (Leak).

The classifier achieved near-perfect performance on the Leak dataset, with an overall accuracy of 0.99. Leak precision was 1.00 and recall was 0.99, indicating that almost all Leak-labelled contracts were correctly identified with minimal false negatives. The Not-Leak class also achieved strong precision (0.93) and perfect recall (1.00).

A likely reason for this strong performance is that Leak behaviours manifest in distinctive patterns—such as unbounded transfer functions or liquidity-draining operations—that are less easily confused with the logic of legitimate contracts. These sharp behavioural signatures are well captured by the extracted features, making the class easier to separate compared to more ambiguous categories such as Mint. As a result, Leak detection shows both high sensitivity and high specificity, underscoring the strength of the feature-engineering pipeline in capturing this behaviour.

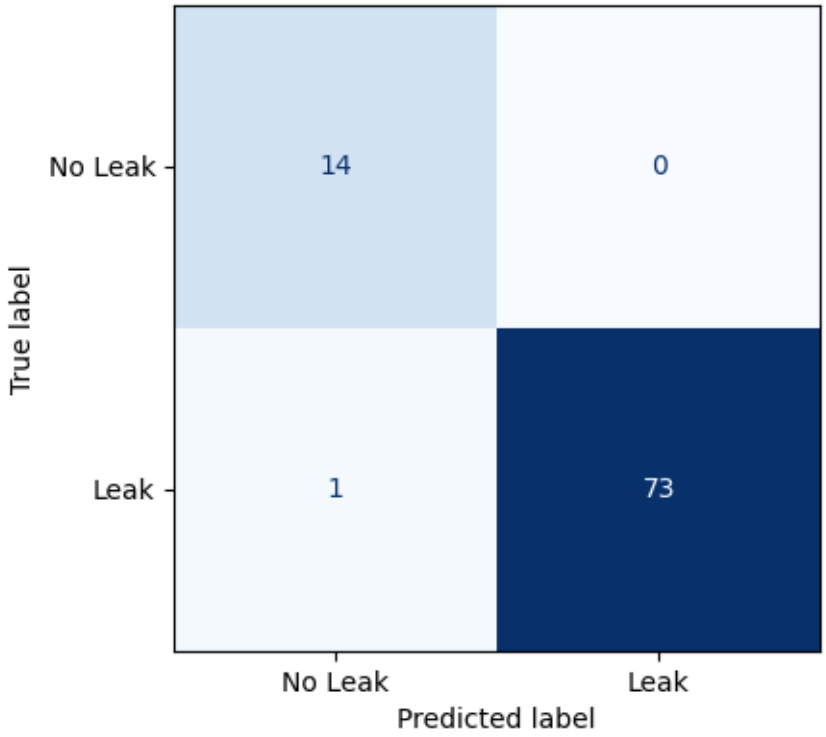

Figure 8: Confusion matrices for Leak predictions on CRPWarner large sample: leak dataset.

4.3.4 CRPWarner Large Sample: Limit

Classification Results

The model was further evaluated on the CRPWarner large sample containing Limit-labelled contracts. Performance metrics are summarised in Table 6, while Figure 9 shows the corresponding confusion matrix.

Class	Precision	Recall	F1-score	Support
No Limit	1.00	0.92	0.96	13
Limit	0.99	1.00	0.99	79
accuracy			0.99	92

Table 6: Classification report on CRPWarner large sample (Limit).

The classifier achieved excellent performance on the Limit dataset, with overall accuracy of 0.99. The Limit class obtained both high precision (0.99) and perfect recall (1.00), while the Not-Limit class achieved perfect precision (1.00) but slightly lower recall (0.92), indicating a small number of false positives.

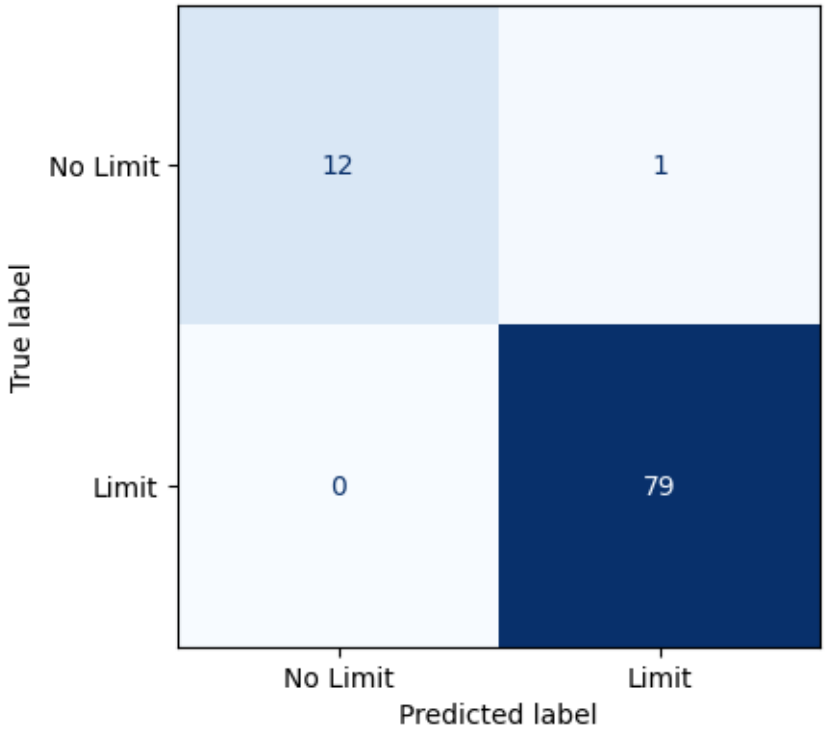

Figure 9: Confusion matrices for Limit predictions on CRPWarner large sample: limit dataset.

This strong performance can be attributed to the fact that Limit behaviours are typically implemented through explicit conditional checks (e.g., caps on transaction size or frequency) that leave distinctive opcode or control-flow signatures. These patterns are relatively easy to separate from the logic of normal contracts, which seldom impose such strict restrictions. The few false positives suggest that some benign contracts may also contain legitimate constraints resembling those of malicious Limit behaviours, leading to occasional over-prediction. Overall, the clarity of Limit patterns within the feature space explains the consistently high precision and recall for this category.

4.3.5 RPHunter

The model was also evaluated on the RPHunter dataset, which covers extended rug pull behaviours beyond those included in CRPWarner. This dataset includes categories such as hidden mint/burn functions, balance modification, restriction logic, and tax manipulation. Performance metrics are reported in Table 7, and the corresponding confusion matrices are shown in Figure 10.

Classification Results

Class	Precision	Recall	F1-score	Support
Hidden Balance Modification	0.67	0.54	0.60	61
Hidden Mint/Burn	0.82	0.59	0.69	379
Address Restrict	0.44	0.71	0.54	195
Amount Restrict	0.39	0.81	0.53	136
Modifiable External Call	1.00	1.00	1.00	1
TimeStamp Restrict	0.08	0.93	0.15	15
Modifiable Tax Address	0.12	0.70	0.21	27
Modifiable Tax Rate	0.86	0.63	0.73	93
micro avg	0.46	0.66	0.54	907
macro avg	0.55	0.74	0.56	907

Table 7: Classification report on RPHunter dataset.

The model achieved mixed performance on the RPHunter dataset. Classes with larger support, such as *Hidden Mint/Burn* and *Modifiable Tax Rate*, showed relatively strong performance (F1 = 0.69 and 0.73, respectively). In contrast, several *minority classes*, for example *TimeStamp Restrict* (15 samples) and *Modifiable Tax Address* (27 samples), exhibited high recall but low precision, resulting in many false positives. The extreme case is *Modifiable External Call*, which contained only a single labelled sample; such limited support prevents any reliable estimate of generalisation and highlights the challenges posed by severe class imbalance.

The overall micro-averaged F1 is 0.54 and the macro-averaged F1 is 0.56, underscoring the difficulty of generalising under severe class imbalance and potential pseudo-labelling noise. To mitigate this, future work can

- rebalance training via class-aware sampling or per-class thresholding,
- add domain-specific features tailored to restriction and tax-manipulation patterns (e.g., control-flow motifs and parameter-range constraints).

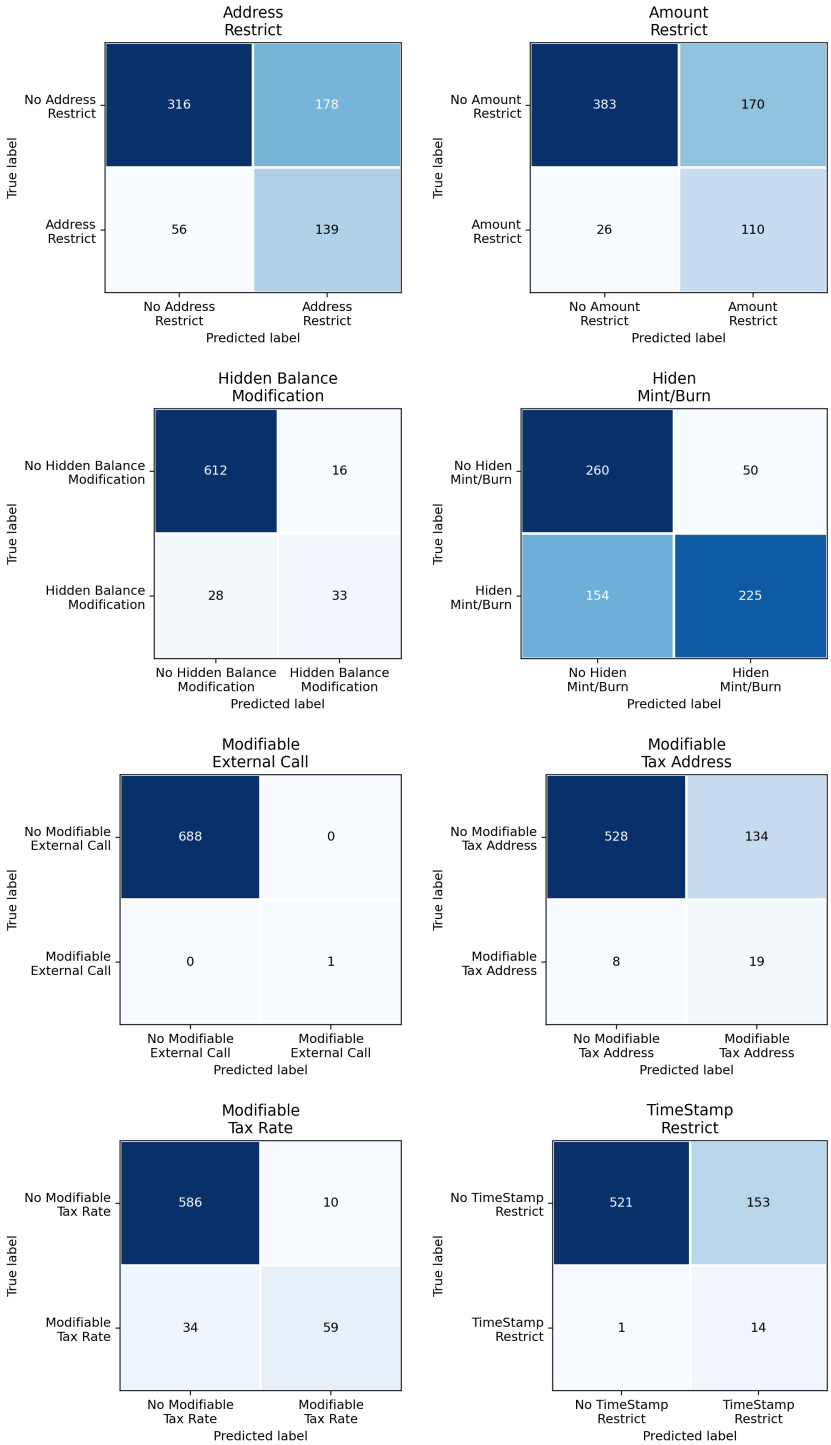

Figure 10: Confusion matrices for predictions on the RPHunter dataset.

4.3.6 Trapdoor

The model was also evaluated on a sample of the Trapdoor dataset, which covers extended rug pull behaviours beyond those included in CRPWarner and RPHunter. Since downloading and preparing the entire dataset of 42,173 contracts was infeasible within the available timeframe, only 2% (approximately 843 contracts) was used for evaluation. This subset nevertheless preserved all key categories, including *Amount Limit*, *Exchange Permission*, *Exchange Suspension*, *Fee Manipulation*, *Invalid Callback*, and *Trapdoor*. Performance metrics are reported in Table 8, and the corresponding confusion matrices are shown in Figure 11.

Classification Results

Class	**Precision**	**Recall**	**F1-score**	**Support**
Amount Limit	0.96	0.98	0.97	370
Exchange Permission	0.88	0.99	0.93	416
Exchange Suspension	0.97	0.92	0.95	374
Fee Manipulation	0.98	0.98	0.98	370
Invalid Callback	0.78	1.00	0.88	7
Trapdoor	0.88	0.97	0.93	472
micro avg	0.93	0.97	0.95	2009
macro avg	0.91	0.97	0.94	2009

Table 8: Classification report on a 2% sample of the Trapdoor dataset.

The classifier achieved consistently strong results on the Trapdoor dataset, with a macro-averaged F1 of 0.94 and a micro-averaged F1 of 0.95. Major classes such as *Amount Limit*, *Exchange Permission*, *Exchange Suspension*, and *Fee Manipulation* all scored above 0.93 F1, while the *Trapdoor* category itself was also classified robustly (F1 = 0.93). The only comparatively weaker class was *Invalid Callback*, which achieved perfect recall (1.00) but lower precision (0.78) due to its very small support size (7 samples). This highlights the persistent challenge of evaluating rare behaviours, where a handful of false positives can substantially reduce precision.

Compared to CRPWarner, which is smaller and focuses on three narrow behaviours (*Mint*, *Leak*, and *Limit*), Trapdoor introduces a richer taxonomy of scam mechanisms,

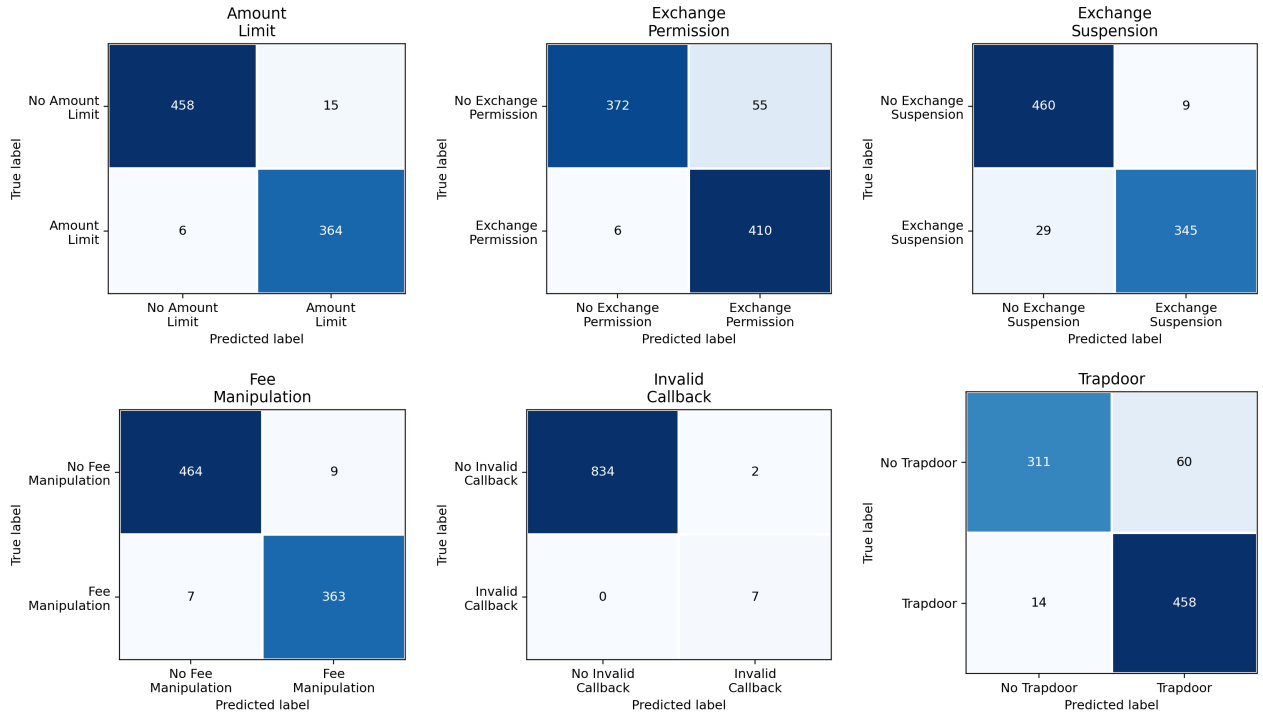

Figure 11: Confusion matrices for predictions on a 2% sample of the Trapdoor dataset.

including permission manipulation, fee control, and callback abuse. Importantly, Trapdoor provides larger class sizes for most behaviours, which reduces the severe imbalance observed in CRPWarner and RPHunter. This broader and more balanced coverage likely contributed to the higher and more stable F1 scores observed.

Overall, these results suggest that the model not only preserves accuracy across datasets but also generalises effectively to the wider behavioural space captured in Trapdoor, supporting its ability to adapt beyond the more limited CRPWarner taxonomy.

4.3.7 Prediction Runtime on Benchmark Datasets

In addition to classification metrics, the elapsed time for prediction was measured across the baseline datasets. All evaluations were run in a CPU-only environment (see Section 4.1.1) with the deployed model served via the backend API. The results are summarised Table 9.

Prediction time varied substantially depending on dataset size and whether Solidity source code, bytecode, or transaction data had to be loaded. For smaller CRPWarner subsets, predictions completed within 200–300 seconds, which is acceptable for interactive

Dataset	Elapsed (s)	Notes
CRPWarner (ground truth)	213.62	no source files
CRPWarner (large Mint)	288.84	no source files
CRPWarner (large Leak)	315.92	no source files
CRPWarner (large Limit)	304.76	no source files
RPHunter	14744.69	no source files
Trapdoor (2% sample)	2414.33	includes loaded source; a 5% sample was attempted but exceeded 18 hours runtime on CPU-only hardware

Table 9: Prediction runtime on benchmark datasets.

analysis. In contrast, the RPHunter dataset required nearly 4.1 hours (14,744 seconds), reflecting its larger size and the increased complexity of evaluating multiple behaviour labels. The Trapdoor evaluation was limited to only 2% of the dataset due to runtime constraints, requiring approximately 40 minutes (2,414 seconds). Extrapolated to the full dataset, this would have taken close to a full day of processing on the available CPU-only hardware, operating in a single thread without caching of predictions.

To improve runtime efficiency in practical deployments, the system could implement caching mechanisms to store both raw input data and the results of previous predictions. This would avoid repeated parsing of Solidity source code, bytecode, or transaction logs, and would also allow contracts that have already been analysed to be skipped or retrieved instantly from the cache. Such a design would substantially reduce latency in interactive settings and enable more scalable batch evaluations, particularly when combined with GPU acceleration or distributed processing.

4.3.8 Cross-Dataset Comparison

Across the three benchmark datasets, the mixed model demonstrated strong predictive performance but also highlighted distinct challenges:

- **CRPWarner:** High accuracy across Mint, Leak, and Limit, though Mint precision was reduced due to over-prediction.
- **RPHunter:** Moderate generalisation (macro F1 = 0.56), with severe class imbalance limiting precision for rare behaviours such as TimeStamp Restrict and

Modifiable Tax Address.

- **Trapdoor (2% sample):** Strong performance overall (macro F1 = 0.94), with only minority classes like Invalid Callback showing instability due to extremely low support.

These results indicate that the model generalises well across diverse scam taxonomies, but that dataset imbalance and computational scalability remain critical issues for real-world deployment.

Chapter 5

Conclusions

5.1 Summary of Findings

This dissertation set out to investigate whether rug pull scams in Decentralised Finance can be detected and categorised automatically using machine learning, even when verified source code is unavailable. To address this challenge, a scalable framework was developed that integrates transaction-level, bytecode, and sequential features, with supervised classifiers, anomaly detection, and late-fusion ensembles optimised via Optuna.

The framework was evaluated across benchmark datasets, starting with the CRPWarner ground truth, and extending to the larger CRPWarner samples, RPHunter, and Trapdoor datasets. On CRPWarner, the system achieved consistently strong classification performance across Mint, Leak, and Limit behaviours (macro F1 $\approx$ 0.91). On the larger and more imbalanced RPHunter dataset, performance was moderate (macro F1 $\approx$ 0.56), reflecting the difficulty of detecting rare behaviours such as Timestamp Restrict or Modifiable Tax Address. On the Trapdoor dataset, even with only 2% of the contracts tested, the model generalised effectively (macro F1 $\approx$ 0.94). These results confirm that rug pull behaviours leave detectable patterns in transaction and bytecode features, and that multi-source fusion significantly improves robustness compared with single-modality models.

The anomaly detection module showed limited precision but achieved perfect recall, confirming its value as a fail-safe mechanism for catching suspicious contracts outside the labelled taxonomy. Self-learning, in the form of pseudo-labelling, successfully expanded

the training set, though imbalances persisted and high-confidence thresholds were essential to avoid error amplification. Runtime analysis revealed that scalability remains a practical barrier, with smaller datasets processed in minutes but larger collections requiring several hours to a day on CPU-only hardware.

5.2 Limitations

Several limitations were identified during this work. First, labelled data remains scarce and imbalanced: rare behaviours had weak support, reducing classifier stability. Second, the reliance on CPU-only resources limited the scale and efficiency of experiments, particularly for Trapdoor and RPHunter. Third, while pseudo-labelling expanded datasets, it risked introducing noisy labels if not carefully controlled. Finally, the framework has not yet been validated in real-time settings, where throughput and latency constraints may affect feasibility.

5.3 Future Work

Future work should prioritise balancing datasets through cost-sensitive learning, augmentation, or adversarial data generation to address class imbalance. Extending the framework across chains such as BNB Chain, Polygon, and Solana will test cross-chain generalisation. Incorporating more advanced anomaly detection and clustering could help identify emerging scam types beyond existing taxonomies. Improving runtime scalability through caching, GPU acceleration, and distributed processing will be critical for real-world deployment. Finally, translating the framework into an investor-facing dashboard or alert system would move the research toward practical application, offering real-time scam warnings in decentralised exchanges.

5.4 Final Reflection

This dissertation has demonstrated that automated detection and categorisation of rug pull scams is both feasible and effective using blockchain transaction data and compiled bytecode. By combining supervised learning, anomaly detection, and iterative self-learning in a unified framework, the system moves toward scalable and adaptive fraud detection. Although challenges of imbalance, data scarcity, and runtime efficiency remain, the work provides a strong foundation for further development of blockchain fraud analytics and contributes to improving transparency and trust in the DeFi ecosystem.

Bibliography

[1] C. Sechting and P. Raschke, "A taxonomy of anti-fraud measures within token economy: Insights from rug pull schemes," in *2024 6th Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pp. 1–9, 2024.

[2] R. Dupre, J. Fajtl, V. Argyriou, and P. Remagnino, "Improving dataset volumes and model accuracy with semi-supervised iterative self-learning," *IEEE Transactions on Image Processing*, vol. 29, pp. 4337–4348, 2019.

[3] H. Wu, H. Wang, S. Li, Y. Wu, M. Fan, W. Jin, Y. Zhao, and T. Liu, "Your token becomes worthless: Unveiling rug pull schemes in crypto token via code-and-transaction fusion analysis," *arXiv preprint arXiv:2506.18398*, 2025.

[4] R. SHARMA, "What Is Decentralized Finance (DeFi) and How Does It Work? — investopedia.com." `https://www.investopedia.com/decentralized-finance-defi-5113835#toc-concerns-about-defi`, 2024. [Accessed 28-08-2025].

[5] R. Moody, "Worldwide crypto & nft rug pulls and scams tracker," Aug 2022.

[6] "Introduction — Etherscan — docs.etherscan.io." `https://docs.etherscan.io/etherscan-v2`. [Accessed 24-08-2025].

[7] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[8] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[9] F. Vogelsteller and V. Buterin, "Erc-20 token standard (eip-20)," 2015.

[10] D. Shirley, W. Entriken, J. Evans, and N. Sachs, "Erc-721 non-fungible token standard," 2018.

[11] H. Adams, N. Zinsmeister, and D. R. Salem, "Uniswap v2 core," 2020.

[12] G. Angeris, T. Chitra, *et al.*, "An analysis of uniswap markets," *arXiv preprint arXiv:1911.03380*, 2020.

[13] Tether, "Transparency," 2025. Issuer claims 1:1 reserves; accessed Aug. 29, 2025.

[14] Tether, "Q2 2025 attestation (bdo italia): Financial figures and reserves," 2025. Accessed Aug. 29, 2025.

[15] CoinMarketCap, "Tether (usdt) — price, market cap and live data," 2025. Accessed Aug. 29, 2025.

[16] CoinMarketCap, "Top stablecoin tokens by market capitalization," 2025. Accessed Aug. 29, 2025.

[17] "CFTC Orders Tether and Bitfinex to Pay Fines Totaling $42.5 Million — CFTC — cftc.gov." `https://www.cftc.gov/PressRoom/PressReleases/8450-21`. [Accessed 30-08-2025].

[18] New York State Office of the Attorney General, "Attorney general james ends bitfinex's illegal activities in new york," 2021. Settlement with Bitfinex and Tether; Feb. 23, 2021.

[19] P. Doli and E. Kollwitz, "The role of financial crime in digital asset scams: Lessons from convicted fraudsters," 2025.

[20] Z. Lin, J. Chen, J. Wu, W. Zhang, Y. Wang, and Z. Zheng, "Crpwarner: Warning the risk of contract-related rug pull in defi smart contracts," *IEEE Transactions on Software Engineering*, 2024.

[21] Etherscan, "Top tokens by market capitalization – tether usd (usdt)," 2025. Market cap of USDT recorded at $167.6 billion as of August 2025.

[22] F. Lee, "What Is Logistic Regression? — IBM — ibm.com." https://www.ibm.com/think/topics/logistic-regression, 2025. [Accessed 21-08-2025].

[23] "What Is Random Forest? — IBM — ibm.com." https://www.ibm.com/think/topics/random-forest. [Accessed 21-08-2025].

[24] A. A. Awan, "A Guide to Bagging in Machine Learning: Ensemble Method to Reduce Variance and Improve Accuracy — datacamp.com." https://www.datacamp.com/tutorial/what-bagging-in-machine-learning-a-guide-with-examples, 2023. [Accessed 21-08-2025].

[25] E. R. Eda Kavlakoglu, "What is XGBoost? — IBM — ibm.com." https://www.ibm.com/think/topics/xgboost. [Accessed 21-08-2025].

[26] I. Amit, "XGBoost vs LightGBM — mr-amit.medium.com." https://mr-amit.medium.com/xgboost-vs-lightgbm-b6ca76620156, 2025. [Accessed 21-08-2025].

[27] "MultiOutputClassifier — scikit-learn.org." https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputClassifier.html. [Accessed 21-08-2025].

[28] "CountVectorizer — scikit-learn.org." https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html. [Accessed 21-08-2025].

[29] "TfidfVectorizer — scikit-learn.org." https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html. [Accessed 21-08-2025].

[30] "SimpleImputer — scikit-learn.org." https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html. [Accessed 21-08-2025].

[31] "StandardScaler — scikit-learn.org." https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. [Accessed 21-08-2025].

[32] C. Stryker, "What is a Recurrent Neural Network (RNN)? — IBM — ibm.com." `https://www.ibm.com/think/topics/recurrent-neural-networks`, 2024. [Accessed 21-08-2025].

[33] J. Jiang, "Simple Weighted Average Ensemble — Machine Learning — medium.com." `https://medium.com/analytics-vidhya/simple-weighted-average-ensemble-machine-learning-777824852426`. [Accessed 22-08-2025].

[34] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[35] J. Barnard, "What Is Anomaly Detection? — IBM — ibm.com." `https://www.ibm.com/think/topics/anomaly-detection`, 2023. [Accessed 22-08-2025].

[36] R. Gillespie, "Detecting fraud and other anomalies using isolation forests," in *Proceedings of the SAS Global Forum 2019*, (Cary, NC), SAS Institute Inc., 2019. Paper 3306-2019.

[37] H. Homayouni, S. Ghosh, I. Ray, S. Gondalia, J. Duggan, and M. G. Kahn, "An autocorrelation-based lstm-autoencoder for anomaly detection on time-series data," in *2020 IEEE International Conference on Big Data (Big Data)*, pp. 5068–5077, 2020.

[38] N. Grech, L. Brent, B. Scholz, and Y. Smaragdakis, "Gigahorse: thorough, declarative decompilation of smart contracts," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 1176–1186, IEEE, 2019.

[39] "RugPull/dataset/groundtruth at main · CRPWarner/RugPull — github.com." `https://github.com/CRPWarner/RugPull/tree/main/dataset/groundtruth`. [Accessed 03-07-2025].

[40] "RugPull/dataset/large at main · CRPWarner/RugPull — github.com." `https://github.com/CRPWarner/RugPull/tree/main/dataset/large`. [Accessed 03-07-2025].

[41] "trapdoor-scam/verified_dataset.csv at main · bsdp2023/trapdoor-scam — github.com." `https://github.com/bsdp2023/trapdoor-scam/blob/main/verified_dataset.csv`. [Accessed 03-07-2025].

[42] P. D. Huynh, T. De Silva, S. H. Dau, X. Li, I. Gondal, and E. Viterbo, "From programming bugs to multimillion-dollar scams: An analysis of trapdoor tokens on decentralized exchanges," *arXiv preprint arXiv:2309.04700*, 2023.

[43] `https://etherscan.io/contractsverified`. [Accessed 21-08-2025].

[44] F. Cernera, M. La Morgia, A. Mei, and F. Sassi, "Token spammers, rug pulls, and sniper bots: An analysis of the ecosystem of tokens in ethereum and in the binance smart chain ({{{{{BNB}}}}}}," in *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 3349–3366, 2023.

[45] "RugPull/dataset/groundtruth/groundTruth.xlsx at main · CRPWarner/RugPull — github.com." `https://github.com/CRPWarner/RugPull/blob/main/dataset/groundtruth/groundTruth.xlsx`. [Accessed 24-08-2025].

[46] "RugPull/dataset/large/sample at main · CRPWarner/RugPull — github.com." `https://github.com/CRPWarner/RugPull/tree/main/dataset/large/sample`. [Accessed 24-08-2025].

[47] "GitHub - bsdp2023/trapdoor-scam — github.com." `https://github.com/bsdp2023/trapdoor-scam`. [Accessed 24-08-2025].

[48] "RPHunter dataset." `https://figshare.com/s/e6e7cce0b6574770e7ce`. [Accessed 04-07-2025].

[49] "Api endpoint list – Etherscan." `https://forms.blockscan.com/public/grid/3E9QiN0ONLhCQVibiP3Z-Bpqhmd7zGXsgapEKJupxiI`. [Accessed 24-08-2025].

[50] "LogisticRegression — scikit-learn.org." `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`. [Accessed 25-08-2025].

[51] "RandomForestClassifier — scikit-learn.org." https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [Accessed 25-08-2025].

[52] "BaggingClassifier — scikit-learn.org." https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html. [Accessed 25-08-2025].

[53] "MLPClassifier — scikit-learn.org." https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html. [Accessed 25-08-2025].

[54] "Python API Reference xgboost 3.0.4 documentation — xgboost.readthedocs.io." https://xgboost.readthedocs.io/en/stable/python/python_api.html. [Accessed 25-08-2025].

[55] "lightgbm.LGBMClassifier LightGBM 4.6.0.99 documentation — lightgbm.readthedocs.io." https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html. [Accessed 25-08-2025].

[56] "IsolationForest — scikit-learn.org." https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html. [Accessed 25-08-2025].

[57] "rugpull-detection-msc-kent-202 — git.cs.kent.ac.uk." https://git.cs.kent.ac.uk/nt375/rugpull-detection-msc-kent-2025. [Accessed 25-08-2025].

[58] "GitHub - patorsiang/rugpull-detection-msc-kent-2025 — github.com." https://github.com/patorsiang/rugpull-detection-msc-kent-2025. [Accessed 25-08-2025].

Appendix A

System Snapshots

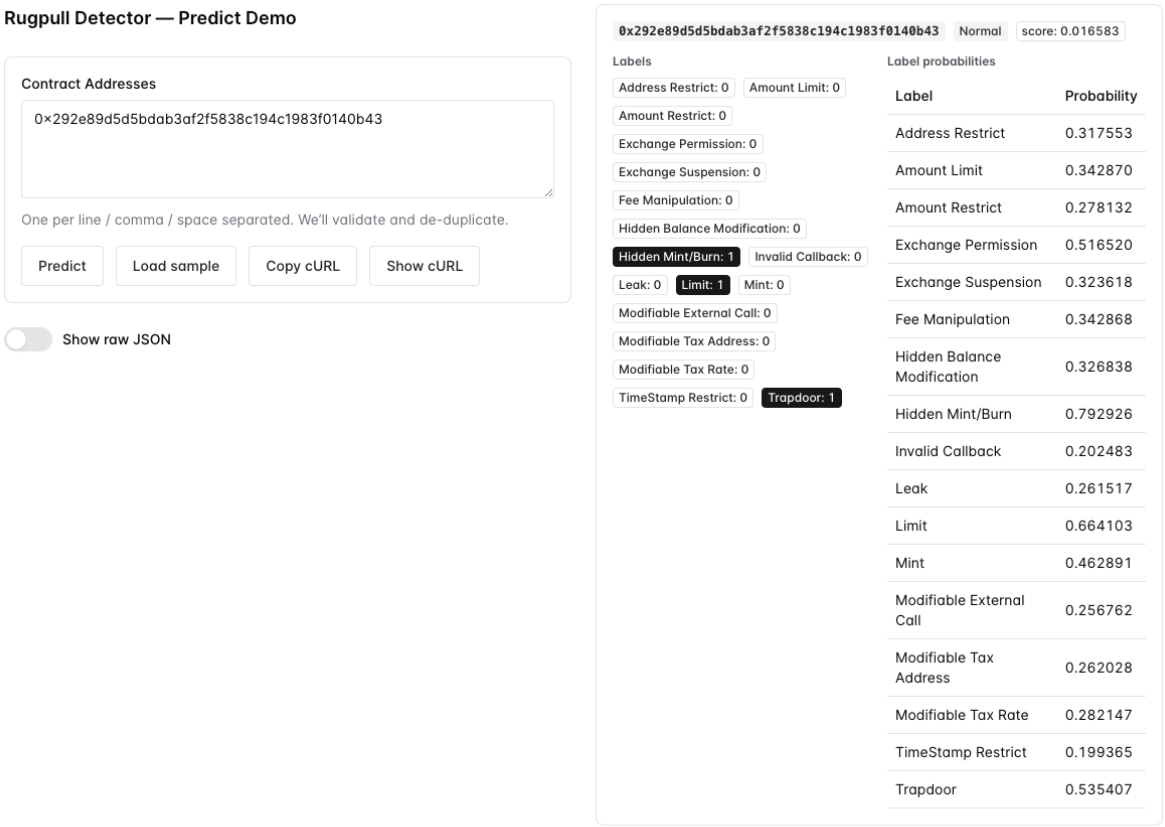

Figure 12: Frontend demo dashboard showing prediction results

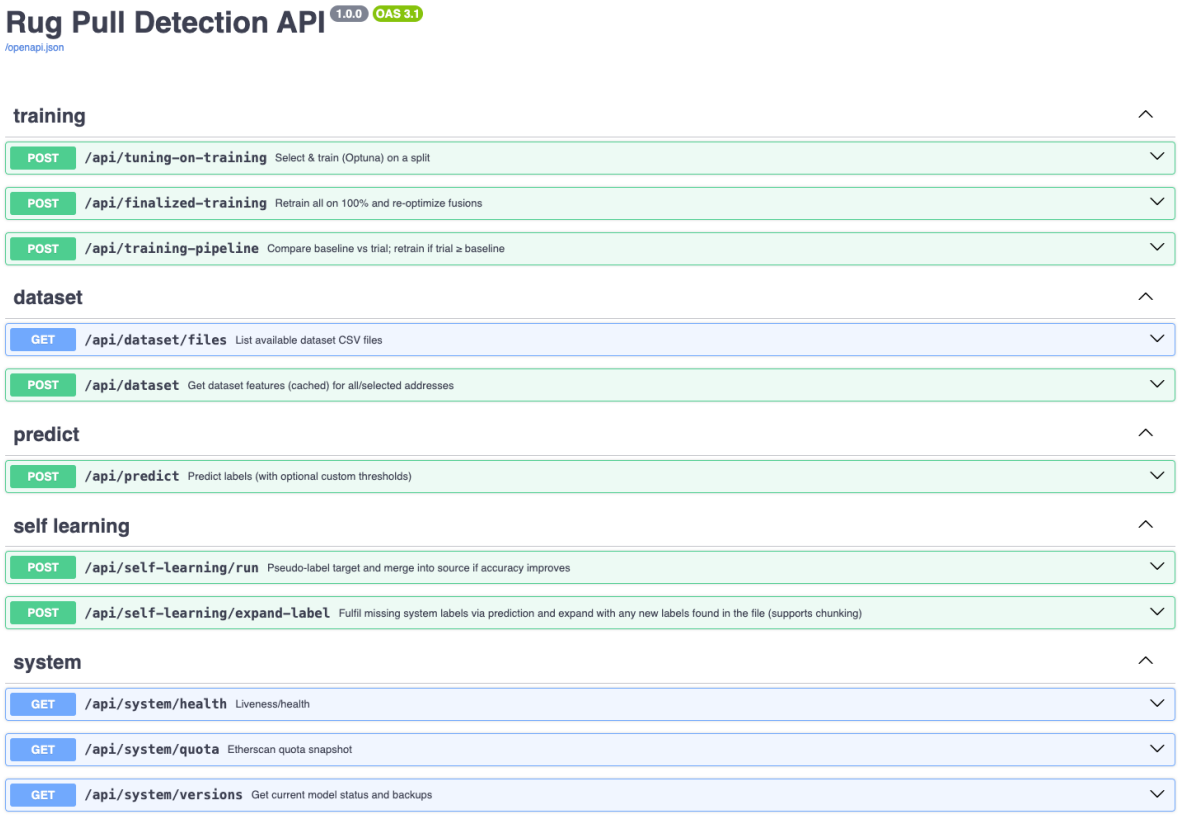

Figure 13: Swagger Docs: show all endpoints in the backend