Multimodal Fusion for Enhanced Smart Contract Reputability Analysis in Blockchain

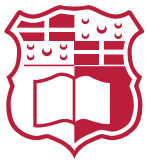
Cyrus Malik

Supervisor: Dr. Josef Bajada

Co-Supervisor: Prof. Joshua Ellul

May 2025

Submitted in partial fulfilment of the requirements for the degree of MSc. in Artificial Intelligence..



L-Università ta' Malta
Faculty of Information &
Communication Technology

Abstract

This study explores the limitations of traditional smart contract reputability assessments, which often rely on either static code analysis or isolated transactional data, missing a comprehensive view of a contract's evolving trustworthiness. Motivated by the need for robust reputability evaluation in blockchain ecosystems, this work introduces a multimodal data fusion framework to integrate static and dynamic data sources. Existing solutions are effective at anomaly detection and vulnerability analysis but fail to combine these data types for holistic insights. Our proposed framework employs boosting algorithms with GAN-based augmentation for opcode embeddings, achieving superior performance in identifying illicit contracts, with a LightGBM model delivering 97.67% accuracy and a recall of 0.942. A CNN-based autoencoder is incorporated for multimodal anomaly detection, effectively identifying abnormal patterns by leveraging the interplay between static code and transactional behavior. The multimodal integration yielded a 7.25% improvement in recall compared to single-source models, confirming its enhanced capacity to detect reputability shifts and abnormal behavior. For long-term monitoring, an LSTM model captures reputability trends, demonstrating low validation loss and minimal prediction lag, ensuring timely and accurate identification of evolving trustworthiness. The results highlight that multimodal fusion significantly enhances predictive accuracy, robust anomaly detection, and the ability to model reputability trends, offering a powerful tool for early risk detection and proactive intervention strategies. This research advances decentralized application security by providing a reliable framework for improving trustworthiness and mitigating potential risks, forming the crux of a sophisticated multimodal data fusion strategy.

Acknowledgements

I would like to express my heartfelt gratitude to my supervisors for their invaluable guidance, insight, and encouragement throughout this research. I am deeply thankful to my family for their unwavering support and belief in me, which provided the strength to persevere. A special thank you to my partner, Jessica, who was patient enough to listen to me talk about this research non-stop and whose love and motivation have been a constant source of inspiration.

Contents

List of Figures

List of Tables

List of Abbreviations

ADASYN Adaptive Synthetic Sampling.

AUC Area Under the Curve.

CAE Convolutional Autoencoder.

CatBoost Category Boosting.

CNN Convolutional Neural Network.

DAO Decentralized Autonomous Organization.

dApps Decentralized Applications.

EVM Ethereum Virtual Machine.

FID Fréchet Inception Distance.

FPR False Positives Rate.

GANs Generative Adversarial Networks.

GBM Gradient Boosting Machines.

GNN Graph Neural Networks.

KDE Kernel Density Estimation.

LightGBM Light Gradient-Boosting Machine.

LSTM Long Short-Term Memory.

MAD Mean Absolute Deviation.

MAE Mean Absolute Error.

MSE Mean Squared Error.

NLP Natural Language Processing.

PCA Principal Component Analysis.

PoS Proof of Stake.

PoW Proof of Work.

RNN Recurrent Neural Network.

ROC Receiver-operating characteristic curve.

SMOTE Synthetic Minority Over-sampling Technique.

t-SNE t-distributed Stochastic Neighbor Embedding.

TPR True Positives Rate.

XGBoost Extreme Gradient Boosting.

1 Introduction

In the rapidly evolving landscape of blockchain technology, smart contracts have emerged as a cornerstone for executing automated, trustless transactions [1]. These protocols, embedded with business logic, are designed to autonomously enforce and verify the terms of agreements [2]. As their adoption grows, ensuring the reliability and integrity of smart contracts has become increasingly critical [3]. A key concern in this context is the reputability of smart contracts — a concept that reflects the perceived trustworthiness of a contract based on its code, behaviour, and context of use.

Assessing reputability is complex. It involves more than verifying code correctness; it also requires understanding a contract's operational history, potential vulnerabilities, and the broader context in which it is deployed [4]. Challenges arise from the dynamic and permissionless nature of blockchain environments, where contracts interact with unknown agents, and new threats emerge over time. While code audits and static analysis tools offer valuable insights [5, 6], they often fail to reflect real-world usage patterns, making reputability a moving target rather than a fixed attribute.

1.1 Problem Definition

Current approaches to evaluating smart contract reputability are limited in scope, often relying solely on embedding-based bytecode analysis or isolated transactional indicators. While effective in identifying code-level vulnerabilities [7], these methods do not capture the broader factors that influence a contract's long-term trustworthiness. For example, transactional data has been used to detect illicit activity [8–10] and highlight reputable projects [11], but such data lacks explanatory depth when used alone.

Reputability is not solely determined by a contract's internal logic or known vulnerabilities. It also depends on how it interacts with users, reacts to external conditions, and adapts over time. Factors such as unusual transaction flows, developer history, patterns of contract deployment, and even responses to governance changes can all impact how trustworthy a contract appears in practice. Traditional analysis frameworks are not equipped to interpret this evolving context.

Moreover, the consequences of misjudging reputability are significant. Inaccurate assessments can lead users to interact with insecure or malicious contracts, potentially resulting in financial losses, exploits, or erosion of trust in decentralized platforms. Conversely, undervaluing reputable contracts may hinder adoption of legitimate innovations. A nuanced assessment method that reflects both static and dynamic dimensions of a contract's behaviour is therefore essential.

1.2 Motivation

The motivation for this research is driven by the growing significance of smart contracts in the blockchain ecosystem, where their reliability and integrity are paramount for ensuring trust and preventing potential risks [12]. As these contracts facilitate various automated processes and transactions, developing a robust framework to assess their reputability—encompassing both their trustworthiness and overall performance is crucial. The goal of this study is to create a comprehensive model that integrates embedding-based bytecode analysis with dynamic transactional data, thereby offering a more nuanced and evolving assessment of smart contract reputability.

While significant research has been devoted to identifying illicit activities and anomaly detection within blockchain networks [13, 14], there is a notable scarcity of studies focused explicitly on reputability. For instance, while research such as that by [8] has advanced our ability to detect suspicious activities and anomalies [14], it has largely overlooked the broader concept of reputability. On the other hand, it has been established that transactional data can help identify reputable projects on EVM-based blockchain platforms [11], it is insufficient for determining the reputability of individual smart contracts in the absence of such transactional evidence. Existing vulnerability analysis methods do play a critical role in identifying security flaws [15], but they predominantly focus on the technical aspects of smart contracts. These analyses are essential for detecting potential vulnerabilities and preventing malicious exploitation, yet they fall short in evaluating the overall reputability or potential illicit nature of a contract.

Additionally, while there is research on temporal aspects of blockchain [16] and malicious account detection through temporal graph properties [17], such studies have primarily concentrated on illicit accounts rather than on the evolution of smart contracts themselves. This focus on account-level analysis does not provide insights into how a smart contract's reputability may evolve over time in response to ongoing interactions and external factors. External factors, in this context, refer to a variety of elements that can influence the performance and trustworthiness of smart contracts beyond their initial code and static metrics. For example, fluctuations in transaction volume, changes in user behaviour, and variations in the types of transactions a smart contract processes can significantly impact its reputability [18].

This research aims to bridge these gaps by integrating static and dynamic data into a unified framework that not only assesses the inherent security and performance of smart contracts but also captures how reputability changes over time. By addressing the limitations of past approaches, this study seeks to offer a more comprehensive understanding of smart contract trustworthiness, thereby enhancing the reliability of blockchain applications and fostering greater confidence in their use.

1.3 Aim and Objectives

This study explores whether smart contract reputability can be inferred from its code and how that reputability shifts over time. As reputability is influenced by both inherent contract logic and contextual behavioural data, a more holistic approach is required. In light of this, the central research question guiding this work is:

> Can a multimodal data fusion framework effectively predict the evolution of smart contract reputability over time by integrating embedding-based bytecode analysis and dynamic transactional data, even in the absence of explicit reputability labels at specific timestamps?

To address this question, the study is guided by the following aim:

> To develop and evaluate a comprehensive framework that integrates embedding-based bytecode analysis with transactional data through multimodal data fusion, enabling reputability prediction and anomaly detection for smart contracts over time.

Achieving this aim requires a structured progression of technical steps. The work is therefore broken down into three key objectives:

1. Acquire, preprocess, and analyse a balanced dataset of smart contracts to evaluate the performance of boosting algorithms in predicting reputability based on bytecode embeddings.
2. Implement multimodal data fusion by integrating bytecode embeddings with transactional data, and apply anomaly detection techniques to detect abnormal patterns in smart contracts over time.
3. Predict how the reputability and anomaly score evolve over time by analysing temporal transaction data using sequence modelling techniques.

While smart contracts are deployed across various blockchain ecosystems [19], this study focuses specifically on platforms that implement the Ethereum Virtual Machine (EVM). This decision is motivated by the widespread adoption of EVM-based chains and Ethereum's foundational role in defining and popularizing the standard. Ethereum's robust developer community, rich ecosystem, and mature infrastructure make it an ideal platform for analysing reputability in an established and representative context.

1.4 Proposed Approach

Our proposed approach is structured into five key stages: data collection, embedding-based bytecode analysis, temporal analysis of reputability based on transaction data, multimodal data fusion, and evaluation of reputability changes over time. Each stage is designed to build upon the previous one, ensuring a comprehensive approach to analyzing smart contract reputability.

The initial stage focuses on collecting a diverse dataset of smart contracts from the Ethereum blockchain. We gather transaction data, source code, and bytecode for both illicit and reputable smart contracts from Etherscan[1]. To address the issue of imbalanced datasets due to the scarcity of illicit contracts, we use data augmentation techniques such as Synthetic Minority Over-sampling Technique (SMOTE), Adaptive Synthetic Sampling (ADASYN), and Generative Adversarial Networks (GANs). We evaluate the performance of our model with each technique to select the best method for balancing the dataset. For embedding-based bytecode analysis, we translate bytecode into a more interpretable format (opcode), simplify it, and extract text embeddings using Word2Vec. Various tree-based models are trained on these embeddings to determine the most effective model for handling the imbalanced data.

In the second stage, we conduct embedding-based bytecode analysis to evaluate the reputability of smart contracts based solely on their source code. This involves extracting various code metrics, including complexity measures and vulnerability assessments using Slither. We use these metrics to create feature vectors that represent the quality and security of the smart contracts. Machine learning models are then trained on these features to predict reputability. The effectiveness of these models is assessed using performance metrics such as accuracy, precision, recall, and F1 score.

Following the embedding-based bytecode analysis, we shift our focus to understanding how the behaviour of smart contracts evolves over time through the use of transaction data. Given the absence of reputability labels at specific timestamps, we employ anomaly detection techniques instead of clustering methods. Features such as transaction volume, participant activity, and transaction frequency are extracted from the transaction history over fixed time windows (e.g., monthly or quarterly). To detect anomalous patterns that might indicate shifts in reputability, we utilize LSTM-based autoencoders and CNN autoencoders, which learn the normal patterns of transactional behaviour and identify deviations from these learned patterns as potential anomalies. The effectiveness of these anomaly detection techniques is evaluated based on reconstruction error, focusing on their ability to detect illicit behaviour and identify significant shifts in the contract's activity over time.

[1] `https://etherscan.io`

The next stage involves integrating embedding-based bytecode analysis with dynamic transaction data to provide a more comprehensive assessment of smart contract behaviour. We develop a multimodal data fusion model, combining features extracted from embedding-based bytecode analysis (such as opcode embeddings and vulnerability features) with transaction-based embeddings derived from the anomaly detection models. This fused dataset allows for a more holistic view of a smart contract's behaviour, capturing both its static properties and evolving dynamics. The multimodal model is used for detecting and predicting reputability and illicit activities over time. To evaluate the efficacy of this approach, we compare the multimodal model's performance with models using only embedding-based bytecode analysis or transaction data, measuring improvements in anomaly detection accuracy, AUC-ROC, and robustness in detecting reputability shifts.

Finally, we assess the temporal evolution of smart contract reputability by applying sequence modelling techniques, specifically LSTM-based models and CNN-Multihead Attention Based LSTM. These models are employed to analyse trends in reputability scores derived from fused data over time, offering a view of how smart contract reputability changes gradually or remains stable. The evaluation focuses on Mean Squared Error (MSE) for trend accuracy and qualitative trend analysis through visual inspection to identify significant shifts in reputability over extended periods.

1.5 Contributions

This thesis makes several key contributions to the field of smart contract analysis and blockchain security:

1. Developed a multimodal data fusion framework that combines embedding-based bytecode analysis with transactional data to enhance smart contract reputability assessment.

2. Applied advanced boosting techniques, specifically LightGBM with data augmentation, for embedded-based bytecode-based reputability prediction, achieving superior performance compared to existing benchmarks.

3. Introduced a multimodal anomaly detection model using CNN-based autoencoders to identify reputability shifts based on both static and dynamic features.

4. Employed temporal sequence models, including LSTM and CNN-LSTM with Multihead Attention, to analyse long-term reputability trends, enabling proactive detection of reputability evolution.

5. Established a comprehensive evaluation framework using quantitative metrics, cross-correlation, and qualitative trend inspection for long-term reputability prediction.

6. Provided two datasets: (1) a labeled dataset of smart contract addresses with their source code and bytee code categorized as illicit or reputable [2] and (2) a dataset of smart contract reputability scores across hourly timestamps, supporting further research in reputability analysis [3].

These contributions offer a robust and holistic approach to smart contract reputability assessment, advancing blockchain security through improved accuracy, timeliness, and practical applicability.

1.6 Document Structure

The remainder of this document covers the following:

Chapter 2 serves as a basis for the background information required to understand the work presented in this study.

Chapter 3 provides a detailed review of previous research covering (but not limited to) illicit activity detection in financial systems, handling class imbalance, hyperparameter optimisation, and handling drifting concepts.

Chapter 4 defines the methodology and the design for the proposed solution.

Chapter 5 Outlines the evaluation framework employed and results, followed by a discussion on the corresponding outcomes.

Chapter 6 includes potential future work stemming from this research and concluding remarks.

[2] https://huggingface.co/datasets/malikcyrus/eth-reputable-illicit-sc-code
[3] https://huggingface.co/datasets/malikcyrus/eth-smart-contract-reputability-hourly-scores

2 Background

This chapter provides the foundational context for this research, covering blockchain technology, smart contracts, and the challenges of assessing their reputability. It reviews embedding-based bytecode analysis methods, anomaly detection techniques, and machine learning approaches such as boosting and temporal sequence modelling. Additionally, it highlights the importance of addressing class imbalance through augmentation techniques to enhance model reliability and evaluation metrics.

2.1 Blockchain - The Backbone of Decentralized Trust

Blockchain technology, often described as the backbone of decentralized trust, represents a transformative innovation in the way digital transactions are conducted and recorded. Introduced by the pseudonymous figure Satoshi Nakamoto in the seminal 2008 white paper, blockchain was initially conceived as the underlying technology for Bitcoin, the first cryptocurrency [20]. However, its potential has since been recognized across various sectors, including finance, retail, healthcare, and more.

At its core, a blockchain is a decentralized, distributed ledger that records transactions in a series of blocks. Each block contains a list of transactions and a cryptographic hash of the previous block, linking them together to form a chain. This structure ensures that once a block is added to the blockchain, the information it contains is immutable, providing a high level of security and trust [21]. The decentralized nature of blockchain means that no single entity has control over the entire network, which is maintained by a distributed network of nodes. These nodes achieve consensus on the validity of transactions through mechanisms like Proof of Work (PoW) or Proof of Stake (PoS), ensuring the integrity and immutability of the blockchain by making it computationally impractical for malicious actors to alter transaction history or double-spend resources. This resilience is achieved by requiring substantial computational effort (in PoW) or economic stake (in PoS) to participate in consensus, thereby deterring dishonest behaviour and maintaining the network's trustworthiness even under adversarial conditions [22, 23].

The transparency and immutability of blockchain technology have led to its adoption in a wide range of applications beyond cryptocurrencies. In supply chain management, for example, blockchain can be used to track the provenance of goods, ensuring transparency and reducing the risk of fraud. In healthcare, blockchain is being explored as a way to securely share patient data between providers while maintaining patient privacy [24]. Despite its many benefits, however, the decentralized and pseudonymous nature of blockchain also presents challenges, particularly when it comes to ensuring the trustworthiness of the participants and the contracts they execute on the network.

2.2 Smart Contracts - Self-Executing Agreements

Building on the foundational principles of blockchain technology, smart contracts are one of the most innovative applications within the blockchain ecosystem. A smart contract is a segment of code designed to automatically execute specified actions based on pre-defined conditions. While it can facilitate agreements between multiple parties, such as buyer-seller transactions, it is not restricted to this function. For instance, it can be employed for personal use cases, such as managing funds or automating personal financial transactions. However, in such scenarios, it does not represent a contract in the conventional sense, as there is no interaction between distinct parties involved.

The code embedded in smart contracts is distributed across a decentralized blockchain network, enabling trusted transactions and agreements among anonymous parties without relying on a central authority, legal system, or external enforcement mechanism [2]. However, this trust depends on the correctness of the code itself and its acceptance by all involved parties. In practice, even if a contract's code is immutable and self-executing, any party can still seek legal recourse or dispute outcomes outside the blockchain, regardless of the enforced code.

The concept of smart contracts was first introduced by Nick Szabo in 1994 [20], well before the creation of blockchain technology, but it was not until the advent of Ethereum in 2015 that the full potential of smart contracts could be realized [25]. Ethereum, a decentralized platform that runs smart contracts, uses a Turing-complete programming language called Solidity, which allows developers to create complex Decentralized Applications (dApps). These smart contracts are executed on the Ethereum Virtual Machine (EVM), which ensures that they run exactly as programmed without any possibility of downtime, censorship, fraud, or third-party interference.

Smart contracts offer several advantages over traditional contracts. They automate the execution of agreements, which reduces the need for intermediaries and significantly lowers transaction costs. Additionally, because the terms of the contract are coded into the blockchain, they are transparent and immutable, providing a high level of trust and security. However, the immutability of blockchain also means that once a smart contract is deployed, it cannot be altered. This can be a double-edged sword; while it ensures that the contract cannot be tampered with, it also means that any bugs or vulnerabilities in the contract code are permanent and can be exploited by malicious actors. A notable example of this is the Decentralized Autonomous Organization (DAO) hack in 2016, where a combination of vulnerabilities in the smart contract code was exploited, resulting in the loss of $50 million worth of Ether [26].

The development and deployment of smart contracts have revolutionized various industries. In finance, for instance, smart contracts are being used to automate processes such as loans, insurance, and payments, reducing the need for intermediaries and

speeding up transaction times. In supply chain management, smart contracts are used to automate and enforce the terms of agreements between suppliers, manufacturers, and retailers, improving efficiency and transparency [27]. However, as smart contracts become more prevalent, the need to ensure their security and trustworthiness becomes increasingly critical. This is where the concept of reputability comes into play.

2.2.1 Reputability Analysis of Smart Contracts

While blockchain technology has enabled the creation and deployment of smart contracts, it has also introduced new challenges in ensuring the trustworthiness and reputability of these contracts. Reputability in the context of smart contracts refers to the degree to which a contract can be trusted to perform its functions as intended, without vulnerabilities or malicious intent. Unlike traditional financial systems, where reputability is often tied to institutional credibility and regulatory oversight, the decentralized and anonymous nature of blockchain requires new methods for assessing reputability [28].

Reputability is a multifaceted concept that encompasses not only the security and reliability of the smart contract code but also the behaviour and performance of the contract over time. A reputable smart contract should be free of vulnerabilities that could be exploited by malicious actors, should execute its functions as intended without errors, and should maintain a consistent track record of reliable performance. However, assessing reputability is a complex task, particularly in a decentralized environment where the identities of the parties involved are often unknown, and where the code, once deployed, is immutable.

To date, most research has focused on detecting illicit activities within blockchain networks, such as money laundering or fraud [8, 9, 29, 30]. These efforts are typically centred around analysing transaction patterns and identifying anomalous behaviour that may indicate malicious activity. Illicit contracts refer to smart contracts or transactions explicitly designed to facilitate illegal activities, such as money laundering schemes, ransomware payments, or unauthorized access to funds.

In contrast, non-reputable contracts encompass a broader category, including those that may not explicitly violate laws but fail to demonstrate trustworthiness or reliability. This could include contracts with poorly written or buggy code, a lack of operational transparency, or inconsistent delivery on their intended functions. While detecting illicit contracts is an essential aspect of ensuring blockchain security, it is insufficient for assessing the broader concept of reputability, which also includes positive attributes such as adherence to best practices in coding, transparency in operations, and consistency in delivering intended outcomes [31].

One of the main challenges in assessing smart contract reputability is the dynamic nature of blockchain ecosystems. A smart contract that is initially deployed with a high

level of trust may lose its reputability over time if vulnerabilities are discovered or if it becomes associated with fraudulent activities. Conversely, a contract that initially appears suspicious might gain reputability through consistent, reliable performance over time. This dynamic nature of reputability adds another layer of complexity to its assessment, requiring continuous monitoring and analysis of the contract's behaviour and performance within the network.

Current approaches to reputability analysis primarily focus on static code analysis, where tools like Slither are used to identify vulnerabilities in smart contract code [32]. While these tools are invaluable for identifying potential risks before a contract is deployed, they do not account for the evolving nature of reputability. Vulnerabilities may go unnoticed during initial deployment and only become apparent as the contract interacts with other contracts and users within the network. Additionally, static analysis does not consider the historical performance of the contract, which can be a key indicator of its reputability.

To address these challenges, a more comprehensive approach to reputability analysis is needed—one that integrates both static and dynamic analyses. This approach should not only assess the quality of the code but also monitor the contract's interactions and performance over time, considering factors such as the frequency and nature of its transactions, its associations with other contracts and users, and its adherence to best practices in coding and execution. By combining embedding-based bytecode analysis with dynamic transaction analysis, a more holistic understanding of smart contract reputability can be developed, which is crucial for maintaining trust and security in decentralized systems.

2.3 Code Analysis in Smart Contracts

As discussed earlier in Section 2.2, once a smart contract has been deployed on a blockchain, it is immutable; any vulnerabilities or flaws in the code cannot be corrected without deploying a new contract. This immutability [33], while essential for trustless operations, also necessitates thorough pre-deployment scrutiny of the contract's code to mitigate the risks of potential exploits [34]. This is where static code analysis becomes a critical tool in the development and auditing of smart contracts.

2.3.1 Static code analysis

Static code analysis is a method of debugging by examining the code without executing it. For smart contracts, this process is crucial as it allows developers to identify and address security vulnerabilities and logic errors before the contract is deployed. Given

the irreversible nature of smart contract deployment, this pre-emptive analysis is a necessary step in ensuring that contracts perform as intended, without exposing users to risks such as reentrancy attacks, unchecked calls, or integer overflows.

Unlike dynamic analysis, which requires the execution of the contract and is therefore limited to the scenarios tested, static analysis reviews all potential code paths, offering a comprehensive assessment of the contract's security posture. In the context of smart contracts, tools like Slither have been developed to automate and enhance this process. Slither, for instance, analyses the Solidity codebase, identifying vulnerabilities, optimizing code, and ensuring adherence to best practices [32].

Several tools have been developed to assist in the static analysis of smart contracts, with Slither being one of the most widely recognized. Slither operates by parsing Solidity code into an intermediate representation that facilitates the identification of potential vulnerabilities. This includes detecting common issues like reentrancy vulnerabilities, where an external contract is allowed to call back into the original contract, potentially leading to unexpected state changes or funds being drained [35].

Another critical vulnerability that Slither can detect is the presence of unchecked calls. In Solidity, external calls are considered successful by default, even if they fail. This behaviour can be exploited by attackers if proper checks are not implemented, leaving the contract in an inconsistent or compromised state [36]. Moreover, the tool is adept at identifying arithmetic vulnerabilities such as integer overflows and underflows, which can occur when arithmetic operations exceed the maximum or minimum values that a data type can store. These issues have been responsible for several high-profile attacks on Ethereum smart contracts, highlighting the importance of thorough static analysis.

The ability to identify and address these vulnerabilities before deployment is crucial for ensuring that smart contracts maintain their integrity and that users can trust their execution. As the use of smart contracts expands into various industries [37–39], the demand for reliable and robust static analysis tools will only grow, making tools like Slither indispensable in the smart contract development lifecycle.

2.4 Tree Models

Decision trees are fundamental building blocks in machine learning, particularly useful for classification and regression tasks. A decision tree model splits the input space into regions, with each region assigned a constant value. Mathematically, a decision tree can be expressed as follows:

$$f(x) = \sum_{m=1}^{M} \theta_m I(x \in R_m) \tag{2.1}$$

Here, R_m represents each of the M regions into which the input space is divided, θ_m denotes the constant value associated with each region, and $I(x \in R_m)$ is an indicator function that equals 1 if x is in region R_m, and 0 otherwise. This basic structure forms the foundation of decision trees and serves as the basis for more advanced techniques like bagging and boosting. These methods enhance the performance of decision trees by addressing key challenges such as variance and bias.

2.4.1 Boosting: A Sequential Learning Approach

Boosting focuses on improving predictive performance by sequentially training models, where each model corrects the errors of its predecessor, thereby reducing both bias and variance. Each model in the sequence attempts to correct the errors made by its predecessors [40]. In boosting, instances that are misclassified by previous models are given higher weights, thereby focusing the subsequent models on the harder-to-classify examples. The combined model, known as the “strong” classifier, is more accurate than any individual “weak” classifier.

AdaBoost, or Adaptive Boosting, is one of the earliest and most popular boosting algorithms [41]. It combines multiple weak learners, typically decision trees, by iteratively adjusting the weights of misclassified instances, ensuring that subsequent models focus more on the harder-to-classify examples. This approach distinguishes it from Gradient Boosting, which optimizes a loss function by training models on residual errors.

Gradient Boosting Machines (GBMs)

Gradient Boosting Machines (GBM) further refine the boosting approach by applying gradient descent to optimize the loss function iteratively [42]. In GBMs, each new model is trained to minimize the residuals (errors) of the combined ensemble of previous models. The optimization process is guided by the gradient of the loss function, which determines the direction in which to adjust the model parameters.

The flexibility of GBMs allows them to be applied to a wide range of tasks, from regression to classification. However, GBMs can be computationally intensive and prone to overfitting if not properly regularized.

Figure 2.1 illustrates the overall architecture of a Gradient Boosting Machine. As shown, each base classifier is trained on a weighted form of the training set, with weights determined by the prior base classifier. After training, the weak classifiers are combined to form the final classifier known as the ”strong” classifier. The iterative process of boosting allows the model to progressively improve its predictions by focusing on the errors of the previous models.

Several advanced implementations of GBMs have been developed to address the

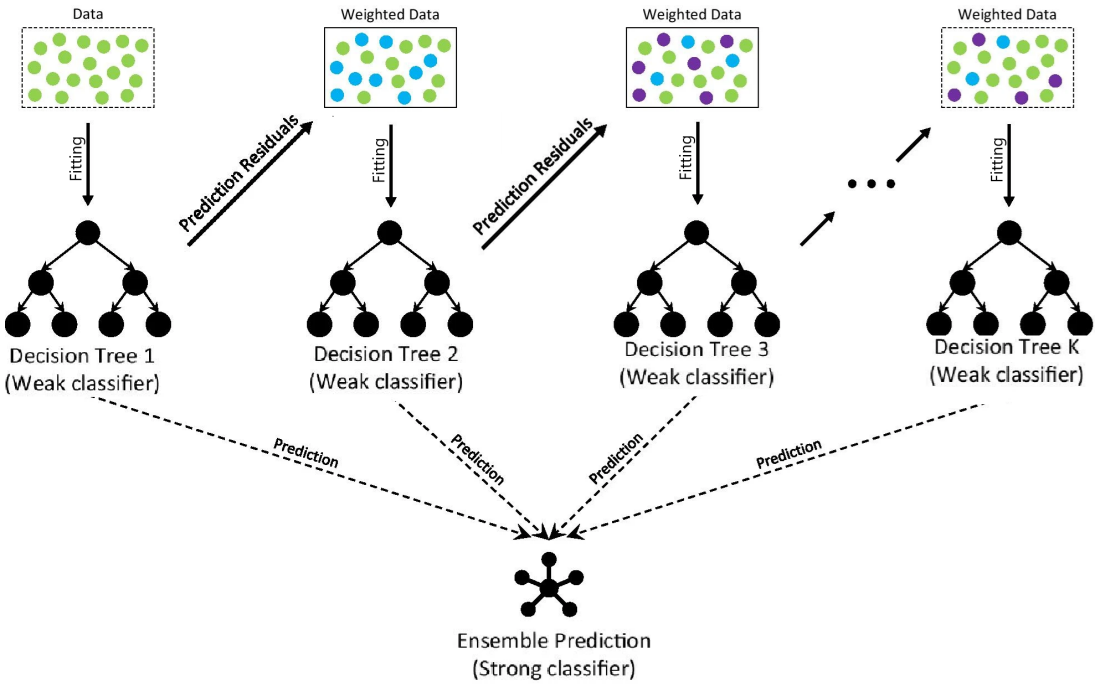


Figure 2.1 Boosting Architecture [43].

limitations of the original algorithm, such as slow training speed, high memory consumption, and lack of scalability for large datasets. These include Light Gradient-Boosting Machine (LightGBM), which improves efficiency and scalability through histogram-based learning and leaf-wise growth; Extreme Gradient Boosting (XGBoost), which enhances regularization and computational speed with optimized tree-building techniques; and Category Boosting (CatBoost), which focuses on reweighing misclassified instances to improve performance.

XGBoost, LightGBM and CatBoost

XGBoost [44], or eXtreme Gradient Boosting, extends the traditional GBM by incorporating additional regularization techniques, which help prevent overfitting [44]. One of its key innovations is the inclusion of a regularization term in the objective function, which balances model complexity with predictive power, thus improving generalization. XGBoost also employs a sophisticated tree pruning mechanism that halts tree construction when further splits no longer improve the model, enhancing both efficiency and effectiveness. Its resilience has led to XGBoost's broad adoption in tasks like anomaly detection in blockchain networks [8].

CatBoost [45], short for Categorical Boosting, is another advanced gradient boosting algorithm specifically designed to handle categorical data efficiently without requiring extensive preprocessing. Unlike XGBoost and LightGBM, which require categorical variables to be encoded manually (e.g., via one-hot or label encoding), CatBoost intro-

duces a novel technique for handling categorical features natively. It uses a process called "ordered boosting," which reduces overfitting by ensuring that each training instance is influenced only by past data points, rather than future ones [45]. This approach improves generalization and ensures unbiased predictions.

Another key advantage of CatBoost is its automatic handling of missing values and its ability to optimize over both CPU and GPU environments efficiently. These features make it particularly effective in domains such as e-commerce, where data is often high-dimensional and contains categorical variables like product categories or user preferences. Furthermore, CatBoost is highly robust to hyperparameter tuning and often requires fewer iterations to achieve strong results compared to other boosting algorithms, making it a practical choice for time-sensitive projects.

LightGBM (Light Gradient Boosting Machine) [46] is engineered to be both highly efficient and scalable, particularly for large datasets with high dimensionality. What sets LightGBM apart from other gradient boosting frameworks are its innovative techniques, such as Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) [46]. GOSS selectively samples data points based on their gradients, prioritizing those with larger gradients to accelerate training without compromising accuracy. This enables the model to focus on the most informative data points, significantly reducing the computational burden. Meanwhile, EFB bundles mutually exclusive features, effectively reducing the number of features processed in each iteration. These characteristics make LightGBM particularly advantageous in scenarios such as finance and cybersecurity, where speed and scalability are critical [11].

2.5 Classification Metrics

To evaluate the effectiveness of a binary classification model, it is essential to use appropriate metrics that provide insights into its performance. These metrics help assess the model's ability to correctly distinguish between the two classes, which in this context are reputable (positive class) and illicit (negative class) smart contracts.

In the case of smart contract analysis, accurate classification is crucial as it allows for the identification of potentially illicit contracts and helps ensure the security and reliability of the blockchain ecosystem. To achieve a comprehensive evaluation of the models used, we employ several key metrics, including accuracy, precision, recall, and F1-score. Each of these metrics offers a unique perspective on model performance, particularly in the presence of class imbalance.

Before detailing the metrics, it is essential to define the core concepts involved in their calculation:

1. **True Positives (TP)**: The number of correctly identified reputable smart contracts.

2. **True Negatives (TN)**: The number of correctly identified illicit smart contracts.
3. **False Positives (FP)**: The number of illicit smart contracts incorrectly identified as reputable.
4. **False Negatives (FN)**: The number of reputable smart contracts incorrectly identified as illicit.

Using these definitions, the evaluation metrics are defined as follows:

1. **Accuracy**: The proportion of correctly classified smart contracts out of the total. It is given by:
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.2}$$
2. **Precision**: The proportion of correctly identified reputable contracts out of all contracts classified as reputable. It is defined as:
$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.3}$$
3. **Recall**: The proportion of correctly identified reputable contracts out of all actual reputable contracts. It is calculated as:
$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.4}$$
4. **F1-Score**: The harmonic mean of precision and recall, which is particularly useful for imbalanced datasets as it takes both false positives and false negatives into account. The F1-Score is expressed by:
$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{2.5}$$

2.6 Data Imbalance and Augmentation Techniques

Data imbalance is a pervasive challenge across various domains, particularly in financial datasets, where certain classes of interest, such as illicit accounts or fraudulent transactions, are significantly under-represented compared to the majority class. This imbalance is commonly observed in datasets involving smart contracts, financial transactions, and cybersecurity, where the number of legitimate entities or activities vastly outnumbers the illicit or fraudulent ones.

Class imbalance can lead to several issues in machine learning models. Primarily, models trained on imbalanced data tend to be biased toward the majority class, which

can result in poor predictive performance for the minority class. This issue is particularly critical in financial domains, where the failure to correctly identify illicit activities, such as money laundering or fraudulent transactions, can have severe consequences [47, 48].

Mathematically, consider a binary classification problem where the dataset consists of N samples, with N_1 samples belonging to the majority class (e.g., legitimate transactions) and N_2 samples belonging to the minority class (e.g., illicit transactions), where $N_1 \gg N_2$. The imbalance ratio R can be defined as:

$$R = \frac{N_1}{N_2} \tag{2.6}$$

A high imbalance ratio R leads to models that are likely to predict the majority class more frequently, reducing the sensitivity to the minority class. This phenomenon is particularly problematic in financial contexts, where detecting rare events, such as fraudulent activities or smart contract exploits, is of paramount importance [47, 49].

The impact of class imbalance on model performance is typically measured using metrics like precision, recall, and the F1-score. In imbalanced datasets, models may achieve high precision but low recall, or vice versa, leading to a poor F1-score. This imbalance can be particularly detrimental in financial applications, where the cost of false negatives (e.g., missed fraud cases) can be extremely high [50]. Conversely, a high number of false positives where legitimate smart contracts are incorrectly flagged as untrustworthy can undermine developer credibility, hinder adoption, and lead to unnecessary scrutiny or delays, posing a significant challenge in practical deployments [51].

One of the techniques used to address the challenges posed by class imbalance is to artificially balance the dataset by increasing the representation of the minority class. Three widely used techniques in this context are Synthetic Minority Over-sampling Technique (SMOTE) [52], Adaptive Synthetic Sampling (ADASYN) [53], and Generative Adversarial Networks (GANs) [54].

2.6.1 Synthetic Minority Over-sampling Technique (SMOTE)

SMOTE [52] is a popular technique that generates synthetic samples by interpolating between existing minority class samples. Specifically, SMOTE selects two or more similar instances from the minority class and generates new synthetic samples along the line segments connecting these instances. The synthetic samples are generated as follows:

$$\text{new\_sample} = x_i + \lambda \times (x_j - x_i) \tag{2.7}$$

where x_i and x_j are two randomly selected instances from the minority class, and λ is a random number between 0 and 1 [52]. By enriching the minority class, SMOTE helps balance the training data and reduce bias toward the majority class [55].

2.6.2 Adaptive Synthetic Sampling (ADASYN)

ADASYN [53] builds upon SMOTE by generating synthetic data adaptively, focusing on the minority class samples that are harder to learn. This is achieved by adapting the number of synthetic samples generated for each minority class sample based on the difficulty level determined by the learning algorithm. The adaptation process is driven by the ratio of minority class samples in the k-nearest neighbours:

$$G_i = \left(\frac{\Delta_i}{m}\right) \times G \tag{2.8}$$

where G_i is the number of synthetic samples for minority sample x_i, Δ_i is the ratio of majority samples among the k-nearest neighbours of x_i, m is the total number of minority samples, and G is the total number of synthetic samples [53].

To generate synthetic samples for a given minority class sample x_i, ADASYN selects one of its k-nearest neighbours at random. A synthetic sample is then created as a convex combination of x_i and the selected neighbour x_j:

$$x_{\text{new}} = x_i + \lambda \cdot (x_j - x_i) \tag{2.9}$$

where $\lambda \sim \text{Uniform}(0, 1)$. This process ensures that the synthetic samples are distributed along the line segments connecting x_i and its neighbours, maintaining the original distribution of the minority class.

By focusing on harder-to-learn instances, ADASYN improves the model's ability to generalize, particularly in complex financial datasets [56].

2.6.3 Generative Adversarial Networks (GANs)

GANs [54] represent a significant advancement in the field of data augmentation, particularly for addressing the challenges posed by highly imbalanced datasets. Unlike traditional oversampling methods such as SMOTE and ADASYN, which generate synthetic data by interpolating between existing data points, GANs use a more sophisticated approach that can capture the underlying distribution of the data more accurately, producing realistic and diverse synthetic samples.

As shown in Figure 2.2, the architecture of a GAN consists of two neural networks, the generator and the discriminator, which are trained simultaneously in a competitive setting. The generator's goal is to produce synthetic data that is indistinguishable from real data, while the discriminator attempts to distinguish between real and synthetic data. This interaction can be formulated as a minimax game, where the generator G and discriminator D are optimized using the following value function:

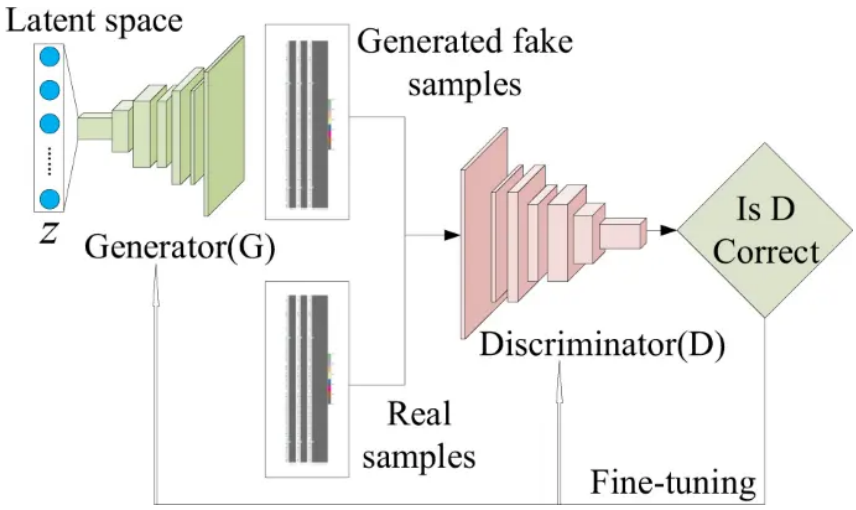


Figure 2.2 Architecture of a GAN showing the interaction between the generator and discriminator networks. The generator (G) produces synthetic data, while the discriminator (D) distinguishes between real and fake data, thereby refining the generator's output. [57]

$$V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.10}$$

Here, x represents real data samples, z is the noise input to the generator, and $p_{\text{data}}(x)$ is the distribution of the real data. The generator G learns to map the noise distribution $p_z(z)$ to the data distribution $p_{\text{data}}(x)$, creating realistic synthetic data points [54]. The GAN training process then seeks the solution to the following minimax optimization problem:

$$\min_G \max_D V(D,G) \tag{2.11}$$

In the context of financial data and smart contract analysis, GANs are particularly advantageous because they can generate synthetic data that closely resembles the minority class, such as illicit transactions or fraudulent activities, which are typically under-represented in the dataset. By doing so, GANs help to mitigate the class imbalance problem, making the trained models more robust and capable of detecting rare but critical events.

One of the key strengths of GANs lies in their ability to produce high-quality, diverse synthetic data that retains the complex correlations present in the real data. This is particularly important in financial datasets, where the relationships between different features can be intricate and where preserving these relationships is crucial for accurate modelling. Unlike simpler augmentation methods that might introduce noise or artifacts into the dataset, GANs are capable of generating realistic data that enhances the model's ability to generalize from the training set to unseen data [58, 59]. Moreover, the flexibility of GANs allows them to be tailored to specific applications within financial modelling.

For instance, in the detection of illicit activities within blockchain networks, GANs can be used to augment datasets by generating synthetic samples of illicit transactions that are difficult to detect. This augmentation process enhances the ability of machine learning models to identify such activities, even in the presence of class imbalance [60].

The effectiveness of GANs in generating realistic data has been demonstrated across various domains, including image synthesis [61], Natural Language Processing (NLP) [62], and financial fraud detection [63–65]. For example, in the context of financial data, GANs have been used to generate synthetic stock market data [66], enabling better risk modelling and fraud detection. In blockchain networks, GANs have shown promise in generating synthetic transaction data that helps improve the detection of anomalies and illicit activities [60, 67].

2.7 Temporal Sequence Modelling

Temporal analysis, or the study of how data evolves over time, is a fundamental aspect of understanding dynamic systems across various domains [68]. In financial systems, temporal analysis allows for the identification of trends, cycles, and anomalies that might indicate significant shifts in behaviour or emerging risks [69]. By examining sequences of data points collected over time, this approach not only enables the detection of patterns signalling changes in market conditions but also provides insights into customer behaviour, as demonstrated in previous studies [70, 71].

In the context of blockchain networks and smart contracts, temporal analysis is particularly crucial for monitoring changes in contract behaviour over time and identifying shifts in reputability. Smart contracts generate a continuous stream of transactional data. This sequential data reflects the ongoing interactions, transactions, and events associated with each contract [72]. Analysing these sequences over time can reveal important insights into the evolving behaviour of contracts, including the identification of normal versus anomalous activity. For example, a sudden increase in transaction volume or a change in the pattern of contract interactions could indicate the onset of fraudulent activities or a shift in contract usage [73].

Temporal modelling [74] of these sequences—whether they originate from smart contracts, stock markets, or other financial systems—enables the development of predictive models that can anticipate future behaviour based on historical data. By leveraging temporal analysis, researchers can build more accurate and reliable systems for monitoring, predicting, and responding to dynamic changes in complex environments.

2.7.1 LSTM Models for Temporal Dependencies

Long Short-Term Memory (LSTM) [75] networks are designed specifically to capture long-term dependencies in sequential data, making them particularly well-suited for temporal modelling tasks such as blockchain transaction analysis. Unlike standard recurrent neural networks (RNNs), LSTMs address the vanishing gradient problem by incorporating memory cells that can selectively retain, update, or discard information over extended sequences. This allows them to identify trends and patterns that evolve over time without losing critical contextual information. In the context of smart contract reputability analysis, LSTMs can learn from historical transaction sequences, enabling them to capture gradual trends or shifts in behaviour that may signify reputability changes.

As shown in Figure 2.3, the core of an LSTM unit includes memory cells and three gates—input, forget, and output—that control the flow of information. At each time step t, the memory cell C_t accumulates information, while the forget gate F_t decides what information from the previous state C_{t-1} should be retained or discarded. This flexibility allows LSTM cells to focus on important temporal dependencies while filtering out noise or irrelevant data, which is crucial for recognizing complex reputability trends in blockchain transactions.

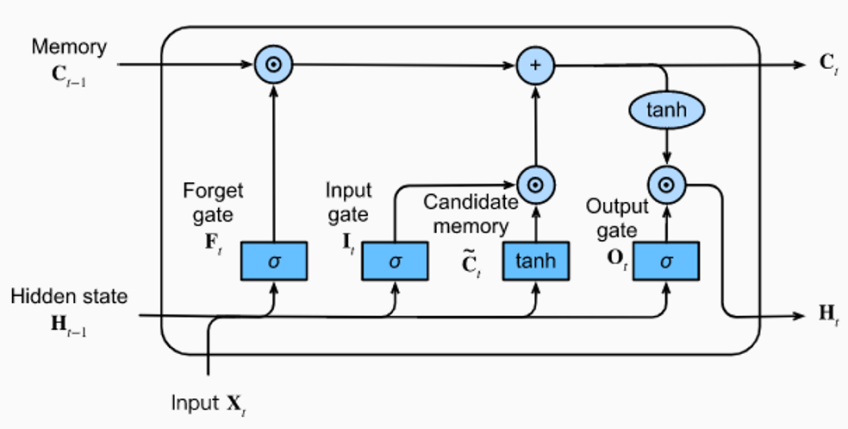


Figure 2.3 The structure of an LSTM cell, featuring the input, forget, and output gates, which control the information flow through memory cells, allowing the model to retain, update, or discard information over time. [75]

In an LSTM-based model, the transaction sequence is encoded into a latent representation that summarizes the historical behaviour of the contract. This latent space, created by the LSTM encoder, captures both recent and long-term interactions, facilitating predictive insights into the contract's future reputability. The hidden state h_t of the LSTM at each timestep retains a representation of past transactions, which allows the model to capture temporal dynamics effectively, as defined in Eq. 2.12.

$$h_t = \sigma(W_x x_t + W_h h_{t-1} + b_h) \tag{2.12}$$

where h_t represents the hidden state at time t, x_t is the input at time t, W_h are the weights, and b_h is the bias term.

In smart contract analysis, the LSTM's memory cells allow the model to detect subtle reputability changes over time, including anomalies that do not immediately indicate fraud but may signal potential risks. The selective memory in LSTM cells is particularly valuable for long-term monitoring, as it enables the model to retain relevant patterns over extended periods, thereby capturing gradual changes that could impact the reputability score. As shown in Figure 2.3, the combined operations of the input, forget, and output gates allow LSTM networks to efficiently manage both short-term fluctuations and long-term trends within transactional data, providing stakeholders with predictive insights to monitor reputability effectively.

2.7.2 CNN-LSTM with Multi-head Attention

The CNN-LSTM with Multi-head Attention model as shown in Figure 2.4 integrates Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks, and a Multi-head Attention mechanism to effectively analyse both local and sequential patterns in transactional data. This architecture is particularly suited for smart contract reputability analysis, where identifying both short-term fluctuations and long-term dependencies is essential.

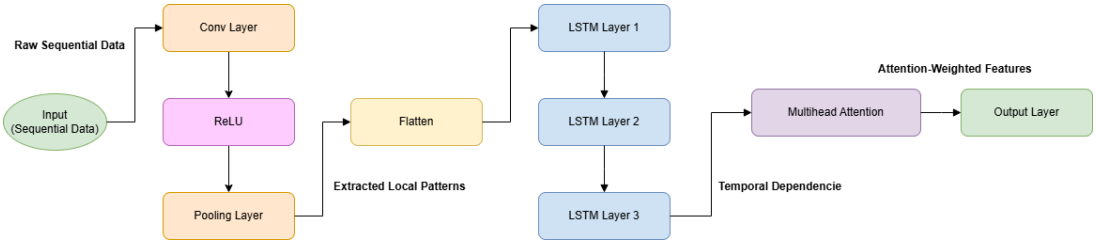


Figure 2.4 The CNN-LSTM with Multi-head Attention architecture. The model begins with convolutional layers to extract local features, followed by LSTM layers that capture sequential dependencies, and concludes with Multi-head Attention to prioritize significant interactions.

The model begins with CNN layers that process the input sequential data to capture localized patterns. These layers apply convolution and pooling operations, which are effective for detecting sudden changes, such as transaction spikes or drops that might indicate shifts in reputability. By using small kernels, the CNN extracts features that highlight important variations within short segments of the sequence, offering granular insights into immediate transactional behaviours.

Following the CNN, the model incorporates a series of LSTM layers that are designed to capture sequential dependencies. s are ideal for temporal analysis, as they retain information across time steps, allowing the model to account for past interactions

when assessing current ones. Here, the layers capture recurring patterns or long-term trends in the transaction data, which could signal gradual changes in a smart contract's behaviour. This aspect of the model is crucial for understanding the trajectory of reputability, as it reflects how a contract's past influences its future interactions.

The Multihead Attention layer further enhances the model's interpretive capacity by dynamically weighing the importance of different points in the transaction sequence. Through multiple parallel attention heads, this mechanism assesses the relevance of each transaction in the context of the entire sequence, emphasizing interactions that are likely to impact reputability more significantly.

For example, an infrequent but large transaction might be given a higher attention weight due to its potential indication of reputability risk, as such transactions can be outliers that signal possible anomalies or vulnerabilities within the blockchain's ecosystem. This increased attention helps to prioritize monitoring and analysis, allowing for more focused assessments. The attention mechanism, defined in Equation 2.13, facilitates this by applying a specific operation that assigns different levels of attention based on the relevance and significance of the input data, aiding in the detection of potential risks more effectively.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.13}$$

where Q, K, and V represent the query, key, and value matrices, respectively, and d_k is the dimension of the keys. By applying this operation across multiple heads, the model captures diverse aspects of the sequence, which aids in recognizing important reputability patterns that might otherwise be overlooked.

The output layer, which follows the attention mechanism, generates a final reputability score or classification, providing an overall assessment of the smart contract's reputability based on the analysed transactional patterns. This combination of CNN, , and Multi-head Attention components enables the model to capture both localized and temporal aspects of the data, making it a powerful tool for real-time reputability analysis in blockchain ecosystems.

2.8 Contract Embeddings

Similar to word embeddings used in NLP, contract embeddings capture relationships between different entities (in this case, smart contracts) based on their contextual behaviour. For example, contracts with similar historical behaviours would have similar embeddings, which helps the model generalize better during the prediction task. Contract embeddings have been successfully used in similar contexts, such as in modelling heterogeneous entities in financial data analysis [76–78]. The use of embeddings allows

the models to differentiate between individual smart contracts and their transactional patterns, ultimately contributing to improved performance.

2.8.1 Anomaly Detection Using Autoencoders and Sequence Models

Anomaly detection in smart contract transactions or blockchain interactions is essential for identifying illicit activities. The combined use of LSTM and Convolutional Neural Network (CNN)-LSTM models, along with autoencoders, provides a versatile toolkit for unsupervised anomaly detection. While autoencoders are well-suited to learn patterns of normal behaviour, sequence models like LSTM and CNN-LSTM with attention mechanisms enable the identification of both gradual and abrupt deviations that could signal fraudulent or risky activities [79, 80].

Given the importance of accurately capturing temporal patterns, advanced machine learning models such as autoencoders have become increasingly relevant. Autoencoders, originally developed for unsupervised learning [81], are highly effective in sequence modelling due to their ability to encode and reconstruct patterns in data, thereby capturing essential temporal dependencies. This structure makes autoencoders well-suited for temporal analysis, where identifying and understanding deviations from typical patterns over time is crucial. Below, we delve into the autoencoder model architecture and its application in temporal sequence modelling.

2.8.2 Autoencoders

Autoencoders [82] are neural networks designed for unsupervised learning, primarily aimed at compressing data into a latent space and then reconstructing it as closely as possible to the original input. The goal is to capture the most important features of the input while filtering out noise or redundant information. In anomaly detection, autoencoders are commonly used to learn patterns from normal behaviour, and deviations in reconstruction can signal anomalies.

Autoencoders consist of two main components:

- **Encoder**: Compresses the input data into a latent representation, reducing dimensionality.
- **Decoder**: Reconstructs the original input from the compressed latent space.

The core idea behind using autoencoders as shown in Figure 2.5 for anomaly detection is that when trained on normal data, the model struggles to reconstruct anomalous inputs, leading to higher reconstruction errors for anomalies. Autoencoders can be adapted to handle both sequential and spatial data, with two prominent variants being LSTM-based and CNN-based architectures.

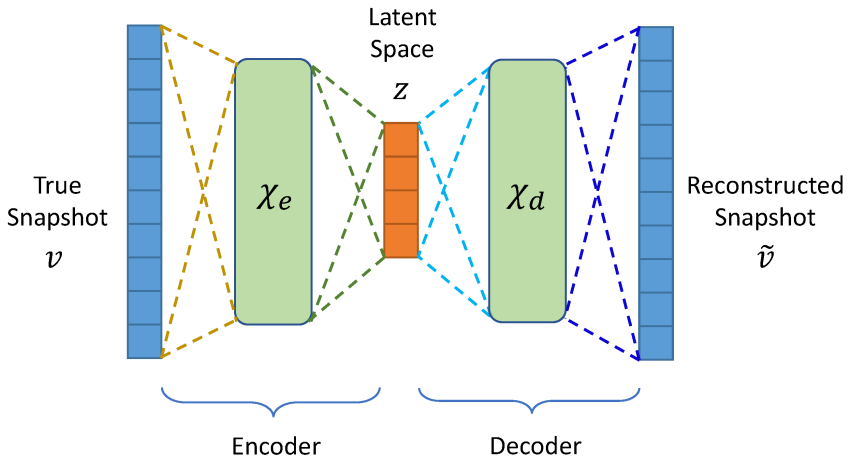


Figure 2.5 The architecture of an autoencoder consisting of the encoder, latent space, and decoder for reconstructing input snapshots from the latent space representation.

LSTM-Based Autoencoders

LSTM-based autoencoders are particularly well-suited for handling sequential data, such as time-series or transaction histories in blockchain systems. LSTM networks are a type of Recurrent Neural Network (RNN) capable of learning long-term dependencies, making them ideal for detecting patterns over time.

The encoder in an LSTM-based autoencoder compresses the sequence of input data (e.g., transactions over time) into a fixed-size latent vector, while the decoder attempts to reconstruct the original sequence from this compressed representation. LSTMs are especially useful in anomaly detection tasks because of their ability to maintain and capture both short-term and long-term dependencies within the data.

In the context of anomaly detection, if the reconstruction error exceeds a certain threshold, the input sequence is flagged as an anomaly, signalling potential illicit activities or unexpected behaviour in a smart contract's transaction history.

Once trained on normal transaction sequences, the LSTM autoencoder is used to evaluate unseen sequences. Anomalous sequences that do not follow the learned normal patterns result in higher reconstruction errors, which can be used to detect fraudulent activities. Reconstruction error is calculated as:

$$\text{Error} = \|x - \hat{x}\|^2 \tag{2.14}$$

Where x is the original input and $\hat{x}$ is the reconstructed sequence from the autoencoder. A high error indicates a deviation from normal behaviour, flagging the transaction sequence as anomalous.

CNN-Based Autoencoders

CNN-based autoencoders leverage convolutional layers to detect anomalies in data that may exhibit spatial patterns, or in cases where local dependencies within the data are important. While CNNs are traditionally associated with image data, they are increasingly applied to time-series or sequence data by treating them as one-dimensional signals.

A convolutional autoencoder replaces the recurrent layers commonly found in sequence models with convolutional layers, which are adept at capturing local patterns or features within the input data. The encoder is composed of successive convolutional layers that progressively downsample the input into a lower-dimensional latent space. Conversely, the decoder reconstructs the original input by upsampling from this latent representation. The convolutional operation performed by the encoder is defined as:

$$h_t = \text{ReLU}(W * x_t + b) \tag{2.15}$$

Where:

W are the convolutional filters,
x_t is the input at time step t,
b is the bias term,
$*$ represents the convolution operation.

CNN-based autoencoders excel at capturing local hierarchies and patterns in the data, which makes them useful for detecting anomalies in shorter time windows or in data where spatial dependencies are critical. In the context of blockchain, CNN-based autoencoders can detect irregularities in batches of transactions or patterns that are too subtle for recurrent models to capture.

Similar to LSTM autoencoders, the CNN-based model is trained on normal data. During the testing phase, any input that produces a high reconstruction error is flagged as anomalous. This architecture is especially effective for identifying local anomalies that may not exhibit strong temporal dependencies but are still critical for detecting fraudulent activities.

2.8.3 Anomaly Detection Using Autoencoders

Anomaly detection in smart contract transactions or blockchain interactions is essential for identifying illicit activities. Both LSTM-based and CNN-based autoencoders provide powerful tools for unsupervised anomaly detection by learning the underlying patterns of normal behaviour and identifying deviations from these patterns [79, 80].

The core principle of anomaly detection with autoencoders relies on reconstruc-

tion error. After training on normal transaction sequences, the autoencoder evaluates new inputs by assessing reconstruction quality. High reconstruction error indicates significant deviation from learned normal behaviour, suggesting anomalies [83, 84].

To classify a sequence as anomalous, a reconstruction error threshold is set, with sequences exceeding it marked as anomalies. This threshold is typically chosen by evaluating model performance on validation data, often using percentile-based methods to identify significant deviations [14, 84]. This approach helps detect irregular or potentially fraudulent activities in smart contract transactions [85].

2.9 Relevance of Multimodal Data Fusion

Multimodal data fusion integrates information from multiple sources to produce a more comprehensive and accurate understanding of a subject [86]. In the context of smart contracts, fusing embedding-based bytecode analysis with temporal transaction data enables a more nuanced and robust analysis. This approach leverages the strengths of each modality, allowing deeper exploration of patterns that may be overlooked when data is examined in isolation [87, 88].

In financial contexts, multimodal fusion enhances analysis by combining complementary data types. Static features, such as code vulnerabilities, highlight structural risks, while transactional data captures dynamic behaviours like spending patterns and anomalies. Fusing these modalities reveals insights not visible when assessed independently [89], enabling the detection of subtle, potentially risky behaviours through a holistic view of both intrinsic and operational characteristics.

Fusing data modalities can be approached through the following techniques:

- **Concatenation:** One of the simplest methods of data fusion involves concatenating the feature vectors from different modalities into a single, unified vector. However, it may not effectively capture the interactions between different data types.
- **Feature-level Fusion:** This technique involves combining features from different modalities before feeding them into the model. Feature-level fusion can involve more sophisticated methods, such as the use of attention mechanisms that weigh the importance of features from each modality differently. This allows the model to focus on the most relevant information from each data type, leading to more accurate predictions [90].
- **Model-level Fusion:** In model-level fusion, each modality is modelled independently, and their outputs are combined using ensemble methods like voting or averaging, allowing each model to specialize in its domain [91].

3 Literature Review

The literature surrounding blockchain technology and smart contracts is rapidly expanding, reflecting the growing importance of these innovations in various sectors, particularly finance and cybersecurity. This chapter reviews key research that informs the analysis of smart contract security and reputability, with a focus on static code analysis, temporal transaction modelling, and multimodal data fusion.

3.1 Smart Contract Reputability

The reputability of smart contracts plays a pivotal role in establishing trust and promoting adoption in blockchain ecosystems. Reputability, in this context, refers to the likelihood of a smart contract behaving reliably and ethically over time.

Current research focuses on methods to evaluate reputability through static and dynamic analyses. Static code analysis examines the syntactic and structural features of smart contracts, such as code complexity, modularity, and adherence to best practices [92, 93]. These features are then correlated with reputability metrics derived from user feedback, transaction history, or external validation mechanisms. For instance, frameworks like SmartEmbed [94] leverage code embeddings to quantify reputability based on code patterns.

Dynamic analysis incorporates runtime data, including transaction frequencies, gas usage, and interactions with other contracts. These dynamic traits are critical for detecting malicious behaviours that may not be apparent from static analysis alone [43]. Combining static and dynamic analyses provides a more comprehensive evaluation of contract reputability.

Transfer learning has emerged as a powerful technique for enhancing reputability classification. Models pre-trained on general datasets are fine-tuned on smart contract data to improve prediction accuracy [95]. This method is especially useful given the limited availability of large, labelled datasets for training robust models. Transfer learning allows models to apply knowledge acquired from general datasets to more specialized tasks, significantly improving prediction accuracy [95]. Reputation Oracles [96] demonstrate how transfer learning can be used effectively to classify smart contracts based on reputability. These systems employ pre-trained deep learning models that are specifically adapted to analyse smart contract bytecode. Bytecode analysis enables more granular detection of patterns and anomalies, facilitating a deeper understanding of potential reputability issues. This method conserves computational resources and enhances model efficiency, addressing the challenge of building robust models without vast labelled datasets.

3.2 Vulnerability Detection in Smart Contracts

The security of smart contracts is paramount to ensuring trust and fostering adoption within blockchain ecosystems. Vulnerabilities in smart contract code, such as reentrancy attacks, integer overflows, and unchecked call values, can have severe consequences. Exploiting these vulnerabilities can lead to significant financial losses and damage to reputability. For instance, the infamous DAO attack in June 2016 [1] led to the theft of $70 million USD, underscoring the catastrophic impact that exploited vulnerabilities can have on trust and security in blockchain networks. Similarly, the 51% attack on Ethereum Classic in January 2019, [2] where malicious actors gained control over the network to manipulate transactions and commit double-spending, exemplifies the need for proactive vulnerability detection and mitigation strategies to prevent such exploits.

Detection of vulnerabilities can be approached through static and dynamic analysis. Static analysis examines the source code without executing it, searching for coding flaws and vulnerabilities through pattern matching and rule-based systems. Tools such as Slither [32] scan smart contract code to identify potential issues such as reentry and overflow errors. Although effective, static analysis has limitations when dealing with new or complex vulnerabilities that are not represented in its rules. This can make it less suitable for identifying new exploit strategies that can be used by malicious actors to compromise the reputability of contracts [92, 93].

Dynamic analysis takes a different approach by executing the code in a controlled environment to observe its behaviour during interactions and transactions. This method can detect issues that arise only under specific runtime conditions, such as gas limit problems or unexpected contract interactions. Although more resource-intensive, dynamic analysis provides insights that static analysis may miss, capturing vulnerabilities that emerge only during transaction execution. The combination of static and dynamic analysis offers a more holistic detection method, helping protect against exploits that could lead to financial theft [97, 98].

Sequence learning models, such as LSTM networks, further extend the vulnerability detection capabilities by analysing opcode sequences extracted from smart contracts. The authors in [43] showed that LSTMs can achieve high accuracy in detecting vulnerabilities, outperforming traditional symbolic analysis tools like Maian. This model benefits from its ability to learn long-term dependencies in sequential data, allowing it to identify complex patterns that may indicate potential security threats [98]. The demonstrated accuracy of the LSTM models, which achieved 99.57% in the tests compared to Maian's 89%, underscores their potential for superior vulnerability detection. This significant improvement indicates that LSTM networks are well equipped to identify complex

[1] https://www.gemini.com/cryptopedia/the-dao-hack-makerdao
[2] https://neptunemutual.com/blog/ethereum-classic-51-attacks/

and novel vulnerabilities that traditional tools may overlook. Such enhanced detection capabilities are vital for protecting smart contracts from exploits that can erode trust and pave the way for illicit activities, thereby safeguarding the overall security and reputability of the blockchain ecosystem.

The integration of multiple data sources can further enhance detection accuracy. Deep learning techniques, such as word embeddings and graph convolutional neural networks, have shown promise in learning from smart contract features to identify vulnerabilities [99]. However, many of these methods tend to focus on single data representations, limiting their ability to fully capture the comprehensive information available. A study by [43] introduced a multimodal decision fusion technique that synthesizes multiple data representations—source code (SC), operation code (OP), and control-flow graphs (CFG)—to improve vulnerability detection. The proposed scheme operates in four stages: modal generation, feature extraction, subclassifier training, and decision fusion. This method transforms smart contract source code into different modalities, extracts features, trains individual models for each modality, and integrates the results for a unified vulnerability assessment.

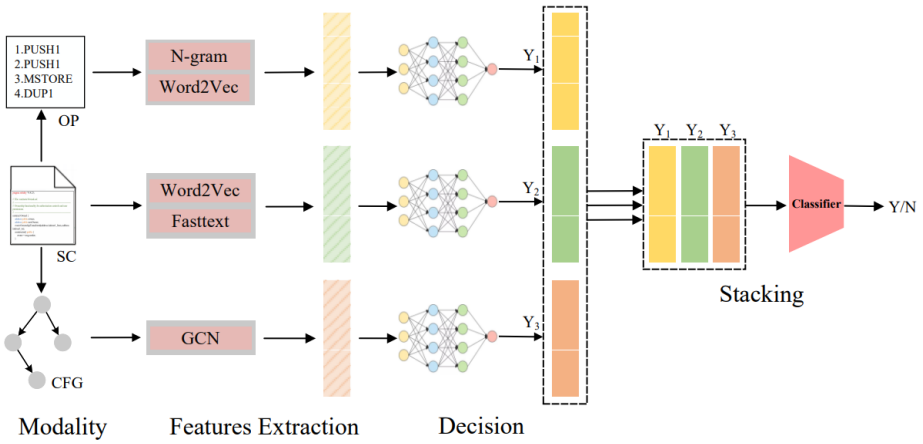


Figure 3.1 Multimodal fusion vulnerability detection scheme graph

As shown in Figure 3.1, the multimodal approach achieved impressive results, with detection accuracies of up to 94.8% and AUC scores surpassing 0.83 for various vulnerabilities, such as arithmetic overflow, re-entrancy, transaction-order dependence, and locked ether. The study also demonstrated that multimodal decision fusion outperforms single-modality methods, indicating that combining different perspectives on the code significantly enhances detection performance. For example, accuracies for arithmetic vulnerabilities were 91.6% (AUC of 0.834), re-entrancy vulnerabilities reached 90.9% (AUC of 0.852), and transaction-ordering dependence had an impressive 94.8% accuracy (AUC of 0.886).

While these techniques are effective, they also come with challenges. The evolution of attack strategies and the continuous development of new vulnerabilities mean

that detection models must be adaptable and capable of real-time learning. Future research should focus on hybrid models that combine static, dynamic, and ML-based approaches to develop adaptable systems capable of maintaining high detection accuracy over time. These systems will play a critical role in identifying potential exploits, supporting the detection of anomalous behaviour that could signal reputability issues, and preventing illicit activities such as unauthorized fund access or network manipulation.

3.3 Illicit Activity Detection

The transparency and immutability of blockchain technology, while fostering trust, also facilitate illicit activities such as fraud and Ponzi schemes. Detecting these activities requires sophisticated methods for analysing transaction patterns and contract behaviours to mitigate financial risks.

Machine learning-based anomaly detection has demonstrated significant success in identifying fraudulent schemes within blockchain ecosystems [100, 101]. These methodologies leverage data-driven techniques to detect anomalous patterns in transaction flows and contract behaviours, crucial for uncovering fraudulent activities such as Ponzi schemes or other illicit practices. A notable example is the work of Bartoletti et al. [102], who utilized temporal transaction modelling to analyse fund distributions and detect irregular patterns indicative of fraudulent contracts. Their approach highlighted the effectiveness of time-based analyses in identifying potential risks within smart contract transactions.

Complementing these approaches, the study by Farrugia et al. [8] specifically examined account-level transactional data to classify Ethereum accounts as either legitimate or illicit using the XGBoost classifier. XGBoost, a gradient-boosting framework, was selected for its scalability, efficiency, and robustness in handling structured and high-dimensional data, making it particularly suitable for blockchain datasets. By analyzing transaction histories, including features such as transaction volumes, frequencies, and timings, the model identified patterns that deviate from typical user behaviour.

The XGBoost model achieved an accuracy exceeding 96%, demonstrating its efficacy in distinguishing between legitimate and malicious activities within the Ethereum network. Additionally, the model exhibited strong precision and recall metrics, minimizing both false positives and false negatives. An important aspect of this study was its focus on feature importance, a capability inherent to XGBoost, which identified key transactional features most indicative of illicit activities. This ability to pinpoint crucial features enhances the model's interpretability and provides deeper insights into the underlying patterns of fraudulent behaviour. These insights provide actionable information for law enforcement and regulatory bodies in combating cryptocurrency-based crimes.

Furthermore, the scalability of the XGBoost model ensures its applicability to large-scale blockchain data, a critical consideration given the rapid growth and complexity of blockchain networks. By enabling the timely processing and analysis of extensive transactional datasets, this approach supports real-time monitoring and detection of illicit activities, which is essential for maintaining trust and security within decentralized ecosystems. These advancements underscore the potential of machine learning to enhance blockchain security by implementing scalable analytical frameworks.

The 51% attack on Ethereum Classic in January 2019[3] underscores the need for proactive illicit activity detection. Such attacks, where malicious entities gain majority control of a network, allow for double spending and manipulation of transactions, resulting in substantial financial losses. Real-world applications, such as monitoring block propagation delays, have shown promise in addressing these risks.

Integrating bytecode embeddings with dynamic transactional data offers a robust approach to identifying illicit patterns. Advances in Graph Neural Networks (GNN) further enhance this capability by modelling complex relationships between entities on the blockchain, improving the detection of sophisticated fraud schemes.

3.3.1 Ponzi Scheme Detection

The detection of Ponzi schemes represents a critical area of focus in addressing blockchain-based fraud. Ponzi schemes exploit the decentralized nature of blockchains and the automated execution of smart contracts to create deceptive investment structures, where returns for early participants are funded by the investments of newer ones. Transaction patterns and operational behaviours are often key indicators of such schemes.

The study [97] addresses the significant problem of detecting Ponzi schemes on the Ethereum blockchain, a pressing issue given the susceptibility of blockchain ecosystems to fraud. Ponzi schemes exploit the blockchain's decentralized nature and the automated execution of smart contracts to create deceptive investment structures, where returns for early participants are funded by the investments of new participants.The main aim of the study [97] was to promote healthier blockchain technology by developing a robust approach for identifying Ponzi schemes, using machine learning methods applied to features derived from contract operations and user account behaviour. This research underscores the importance of fraud detection within decentralized financial platforms by examining the prevalence of Ponzi schemes and their impact on the blockchain ecosystem, echoing concerns raised by prior studies on cryptocurrency scams [101, 103].

To detect Ponzi schemes, [97] combine both code-based and behavioural features in their methodology. They compile the Ethereum smart contract code into bytecode,

[3] https://neptunemutual.com/blog/ethereum-classic-51-attacks/

which is subsequently disassembled into operation codes (opcodes), forming the basis of code-level features for classification. These code-based features are complemented by account-related features derived from transaction data that capture behavioural characteristics indicative of Ponzi schemes. These include metrics such as opcode frequencies, number of investments and payments, known participant ratios, and balance data, which collectively provide a detailed view of each contract's structure and activity.

This methodology draws on established blockchain analysis approaches that use opcode frequencies and transactional data as reliable indicators of contract behaviour [102, 104]. Using a dataset of 1,382 Ethereum contracts, including 131 manually verified Ponzi schemes, [97] employ XGBoost to build a classification model that effectively distinguishes Ponzi contracts based on these features. This dataset, with its manual labels, serves as a reliable ground truth for training and validating the model, enabling a thorough assessment of its effectiveness in detecting Ponzi characteristics.

The results demonstrate the effectiveness of their detection model, achieving an accuracy of 99.56% alongside a recall rate of 96% for Ponzi schemes, underscoring the model's robustness in identifying these contracts. With a precision score of 77%, the model effectively minimizes false positives, achieving an F1-score of 0.86, which combines both recall and precision into a balanced measure of performance. An essential insight from their work is the model's ability to detect Ponzi schemes immediately upon contract deployment, thereby enabling proactive intervention against fraudulent activities on Ethereum [105]. [97] applied their model to the broader Ethereum platform, where it estimated that over 400 Ponzi schemes were active, reflecting the scale of fraudulent activity and the necessity of early detection systems. This proactive approach demonstrates the potential for machine learning to contribute to blockchain security by targeting specific fraud patterns identified through both opcode and behavioural features [92, 93].

In the context of this research, [97] approach provides valuable insights into detecting illicit contracts, particularly those exhibiting Ponzi-like behaviour. However, it is important to note a distinction in how the model classifies contracts, as they categorize contracts as either Ponzi schemes or "normal" contracts, without directly addressing the notion of reputability. In this classification, "normal" contracts encompass all non-Ponzi contracts but do not necessarily indicate trustworthiness or alignment with established ethical standards. In contrast, the concept of reputability used in this study implies a level of trust and reliability that goes beyond merely avoiding fraudulent behaviour [100].

3.4 Summary

This chapter reviewed the state-of-the-art methodologies for analysing smart contract reputability, detecting vulnerabilities, and identifying illicit activities. Key findings and research gaps include:

- Reputability Prediction: Existing models struggle with adaptability to evolving threats; real-time monitoring could enhance prediction accuracy.
- Vulnerability Detection: Many tools face scalability issues and false positives, highlighting the need for integrated approaches.
- Illicit Activity Detection: Effective for known patterns but insufficient for novel schemes; graph-based models show potential.

Our proposed approach directly addresses these challenges by developing a comprehensive framework that integrates embedding-based bytecode analysis and transactional data using multimodal data fusion. This strategy is designed to improve reputability prediction and facilitate the identification of anomalies over time using boosting algorithms, anomaly detection techniques, and sequence modelling.

4 Methodology

This chapter presents a comprehensive explanation of the research methodology employed in this study. By elaborating on the rationale behind our methodological choices, we aim to provide a clear understanding of how our research design addresses the identified research problem. Drawing on the insights from the preceding chapter's critical analysis of existing literature on smart contract reputability analysis within both cryptocurrency and traditional financial systems, we establish a solid foundation for our investigative approach. To elucidate our research process, Figure 4.1 first shows our research pipeline. This visual representation outlines the stages of our approach, starting from data collection and preprocessing, progressing through embedding-based bytecode analysis and multimodal data fusion, and culminating in temporal prediction and anomaly detection for reputability analysis. Moreover, the datasets are available on HuggingFace (Section 1.5) and the code used for this study is available on GitHub.[1]

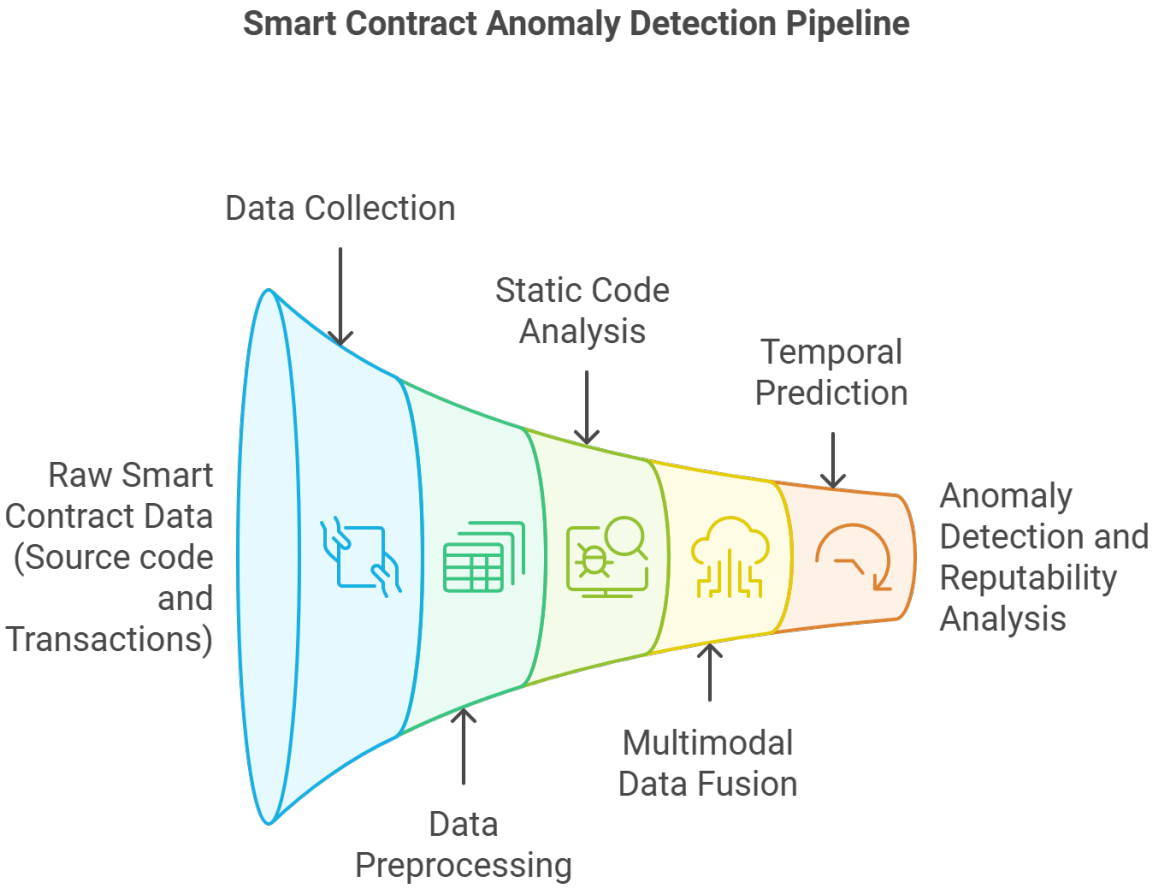


Figure 4.1 Research Pipeline

[1] https://github.com/malikcyrus/crypto-multimodal-reputability

Objective	Experiment	Hypothesis
Objective 1	Experiment 1 (Section 5.1)	*Data augmentation can significantly improve the prediction of illicit smart contracts compared to using an imbalanced dataset.*
Objective 2	Experiment 2 (Section 5.2)	*Fusing bytecode embeddings with transactional data will significantly enhance the accuracy and recall of anomaly detection compared to models that use either data type in isolation.*
Objective 3	Experiment 3 (Section 5.3)	*LSTMs can effectively identify long-term trends in the reputability of smart contracts, offering more reliable insights into reputability evolution compared to short-term anomaly detection.*

Table 4.1 Summary of Objectives, Experiments, and Hypotheses

4.1 Experimental Overview

In the light of conclusions from the literature review, we restate the objectives identified in Section 1.3 and present the relevant experiments that are carried out to address the set objectives in Table 4.1 which also outlines the hypotheses being tested.

The experiments involve both traditional and advanced machine learning techniques, including data augmentation methods, multimodal data fusion, and anomaly detection mechanisms. The outcomes of these experiments will help us understand the role of various data modalities in improving reputability prediction, reducing false positives, and detecting abnormal behaviour over time.

Experiment 1 focuses on addressing class imbalance in the dataset, which is critical for improving both recall and precision in the prediction of illicit contracts. By using oversampling techniques like SMOTE, ADASYN, and GAN-based augmentation, we aim to balance the dataset and enhance the model's ability to accurately detect illicit behaviour while minimizing false positives. Experiment 2 builds on this by integrating transactional data to improve anomaly detection accuracy, precision, and recall. Finally, Experiment 3 explores how reputability evolves over time and how accurately we can predict transitions from reputable to non-reputable states, using temporal transaction data to model changes in smart contract behaviour.

4.2 Datasets

A crucial component of our research is the dataset utilized, which includes both reputable and illicit smart contracts. The pseudonymous nature of cryptocurrencies [20, 25, 106] introduces significant challenges in accurately identifying and labelling smart contracts as either reputable or illicit. This pseudonymity complicates the process of acquiring reliable data on illicit activities, leading to a scarcity of comprehensive datasets essential for assessing reputability.

Furthermore, the sensitive nature of financial data exacerbates this issue, making it difficult to find datasets that provide detailed and reliable information on smart contracts and their transactional behaviour. This scarcity of data not only impacts our ability to analyse and predict smart contract reputability but also hinders the development of robust models for detecting and understanding shifts in reputability over time.

To address these challenges, we sourced data on two primary aspects:

- Reputable Smart Contract Addresses: These were collected from CoinGecko[2], a comprehensive resource for cryptocurrency data. The addresses were obtained through their available dataset.
- Illicit Smart Contract Addresses: These were sourced from CryptoScamDB[3] and a few other documented scams collected by Farrugia et. al [8]. CryptoScamDB provides a repository of known fraudulent smart contracts, crucial for our study.

For our analysis, we required datasets that encompass both the source code and byte code of smart contracts, as well as transaction data for these contracts. To obtain the necessary data, we used the Etherscan API[4], which allows us to fetch:

- Smart Contract Source Code: The human-readable code defining the contract's functionality.
- Smart Contract Byte Code: The compiled code deployed on the blockchain.
- Transactions: Detailed transactional data for each smart contract.

Additionally, while existing datasets such as the Slither Audited Smart Contracts dataset on Hugging Face [107] offer insights into vulnerabilities, they primarily focus on categorizing contracts as vulnerable or not. They do not provide explicit classifications related to reputability or illicitness. Moreover, there is little to no observed correlation

[2] https://www.coingecko.com/en
[3] https://cryptoscamdb.org
[4] https://etherscan.io/apis

between vulnerabilities and the malicious intent of smart contracts [108], suggesting that vulnerability detection alone may not be a reliable indicator of illicit behaviour.

To create a dataset focused on reputability and illicitness, we gathered a total of 3,200 reputable smart contracts and 191 illicit smart contracts.

Since no publicly available dataset exists that specifically addresses code-related features for reputability and illicitness, we curated our own. The following sections provide detailed descriptions of each dataset.

4.2.1 Ethereum Reputable-Illicit Code and Vulnerability Dataset

To obtain data on smart contracts, including both source code and byte code, we utilize the Etherscan API, a comprehensive tool for accessing information about Ethereum-based smart contracts. The Etherscan API provides crucial functionalities that allow us to extract various features related to smart contracts, which are essential for our analysis. Specifically, the API facilitates the retrieval of:

- Smart contract source code; human-readable code that outlines the contract's intended functionality and logic. It is fundamental for understanding the design and potential vulnerabilities within the smart contract.
- Smart contract byte code represents the compiled, machine-readable code that is deployed and executed on the blockchain. Byte code is essential for lower-level analysis and understanding contract operations in a blockchain environment.

The Etherscan API is a crucial resource for our research, enabling us to fetch detailed information on both reputable and illicit smart contracts. This capability is essential for our study, as it provides the comprehensive and up-to-date data required for embedding-based bytecode analysis and vulnerability assessment [15] within the context of reputability and illicit activities.

In addition to utilizing the Etherscan API, we try to incorporate existing datasets to complement our analysis. One such dataset is the Slither Audited Smart Contracts dataset available on Hugging Face [107]. This dataset focuses on vulnerability analysis, categorizing contracts as either vulnerable or non-vulnerable. However, it mainly addresses code vulnerabilities without specific classifications related to reputability or illicitness. Therefore, while it offers valuable insights into vulnerabilities, it does not provide the comprehensive analysis required to assess reputability or illicitness.

To address this gap, we curated a specialized dataset tailored to our focus on smart contract reputability and illicitness. We collected the bytecode of these smart contracts and developed a robust data pipeline to extract features from the opcode sequences. These features capture the intricate behaviours and structural characteristics

of the smart contracts, allowing us to analyse both legitimate and potentially malicious patterns effectively.

4.2.2 Benchmark Dataset: Detecting Ponzi Schemes on Ethereum

For our evaluation on embedding-based bytecode analysis, we use the *Ponzi contract dataset* from [97] on Ponzi Scheme detection (Section 3.3.1). This dataset, accessible from XBlock[5], was selected as a benchmark due to its comprehensive collection of Ethereum smart contracts aimed at identifying Ponzi schemes. This focus on Ponzi schemes aligns with our goal of detecting illicit smart contracts, providing a relevant proxy for reputability.

The dataset includes a total of 3,790 smart contracts labelled as either Ponzi or non-Ponzi, with 200 contracts manually verified as Ponzi schemes and the remaining 3,590 classified as non-Ponzi. This labelling serves as a reliable foundation for training and evaluating models that aim to distinguish fraudulent behaviours within smart contracts. However, while fetching bytecode for these contracts, we encountered limitations: out of the 3,590 non-Ponzi contracts, only 3,569 had accessible bytecode available on Etherscan, as 21 contracts returned "0x" values, indicating unavailable or empty bytecode data.

Moreover, the dataset provides both code-based and behavioural features. The code-based features include opcode frequencies extracted directly from the bytecode, allowing for a static analysis of the contract's underlying structure without requiring transaction history. This static analysis component aligns well with the focus of our research, as it enables an initial assessment of contract behaviour even in cases where transactional data is lacking. In addition to code-based features, the dataset incorporates behavioural data derived from transaction records, such as metrics on participant ratios, the frequency and value of payments, and overall balance activity. These behavioural indicators capture the types of transactional patterns commonly associated with Ponzi schemes and contribute to the robustness of the model's ability to distinguish Ponzi contracts from legitimate ones.

Despite this distinction, including the normal contracts i.e., contracts which are not illicit but also not reputable, from the benchmark dataset [97] remains valuable for evaluating a model's performance in identifying varying types of non-Ponzi contracts. Recognizing normal contracts provides a nuanced view of a model's ability to differentiate between overtly illicit behaviour (e.g., Ponzi schemes) and a broader category of potentially innocuous but non-verified contracts. This approach allows for a more comprehensive assessment of the model's detection capabilities across a realistic spectrum of contract types within the blockchain ecosystem. By examining both Ponzi and normal

[5] `https://xblock.pro/#/dataset/22`

contracts, the study can effectively evaluate the model's ability to detect illicit activity while also discerning contracts that, while not fraudulent, may still lack the indicators of reputability critical for trusted blockchain interactions.

This dataset therefore serves as a valuable benchmark for evaluating the robustness of embedding-based bytecode analysis in detecting anomalies within smart contracts. Through the adaptation of Ponzi-focused data [97], our study examines the performance of reputability detection models on both explicitly illicit contracts and those of ambiguous or unknown trustworthiness, supporting a more comprehensive understanding of reputability and illicitness within the Ethereum ecosystem.

4.2.3 Ethereum Reputable-Illicit Transactions Dataset

To effectively analyse the reputability of smart contracts and understand how it evolves over time, we need a detailed dataset encompassing both the code and transactional data of these contracts. This temporal analysis, combined with code analysis, requires raw transaction data with precise timestamps for accurate monitoring and evaluation.

Existing datasets such as [8] focus on Ethereum transactions but have certain limitations for our purposes. Moreover, the authors in [8] aggregate transactions, which consolidates data into summarized forms rather than providing individual transaction details with timestamps. This aggregation process is not suitable for our temporal analysis, which requires transactions in their raw format to track changes and patterns over time. Additionally, their dataset primarily targets illicit activities by aggregating transactions for normal versus illicit accounts, which does not align with our need for detailed, contract-specific transactional data.

To address these limitations, we collect our own dataset of transactions associated with both illicit and reputable smart contracts. Using the Etherscan API, we retrieved detailed transaction records, which include a variety of attributes essential for our analysis. Table 4.2 outlines the attributes of each transaction returned by an API call used subsequently in our analyses.

This detailed transaction data is crucial for our temporal analysis, allowing us to track and analyse the activity of smart contracts over time. By collecting this data for both reputable and illicit smart contracts, we perform a thorough evaluation of reputability and the evolution of contract behaviour within the Ethereum ecosystem.

4.2.4 Data Pre-Processing

Once the source code and bytecode are collected and vulnerability-related features computed (Section 4.2.1), we preprocess the dataset's bytecode to extract opcodes by dis-

Feature	Description
blockNumber	The block number in which the transaction was included.
timeStamp	The timestamp (in Unix epoch format) indicating when the transaction was processed.
hash	A unique identifier for the transaction.
from	The Ethereum address that initiated the transaction.
to	The Ethereum address that received the transaction. This field can be empty if the transaction involves contract creation.
value	The amount of Ether transferred in the transaction (in Wei).
contractAddress	The Ethereum address of the contract created, if applicable.
input	Data sent to the contract (if applicable). This field is empty for regular transactions.
type	The type of transaction, such as 'call' or 'create'.
gas	The total amount of gas provided for the transaction.
gasUsed	The amount of gas consumed by the transaction.
traceId	A unique identifier for tracing the transaction execution path.
isError	A flag indicating whether the transaction encountered an error (1 for error, 0 for no error).
errCode	An error code if `isError` is 1, detailing the type of error encountered.

Table 4.2 Transaction Attributes Retrieved from Etherscan API

Original Opcode Sequence	Simplified Opcode Sequence
PUSH1 0x80	PUSH
PUSH1 0x40	PUSH
MSTORE	MEMORY
COMPARE	LT
DUP1	DUP
ISZERO	ISZERO
PUSH2 0x10	PUSH
SWAP1	SWAP
PUSH1 0x0	PUSH
DUP1	DUP
REVERT	RETURN

Table 4.3 Example of Original and Simplified Opcode Sequences

assembling the hexadecimal bytecode using the 'pyevmasm' library[6].

To streamline the analysis of these opcode sequences, we simplify the opcode sequence by categorizing each opcode into predefined groups. This categorization reduces the complexity of opcode sequences, facilitating more efficient analysis. We use a set of high-level categories, such as arithmetic operations, comparisons, logical operations, and memory operations. Each opcode is matched against these categories and replaced with a representative category label if it belongs to one of the predefined groups [109]. This approach helps in abstracting the opcode sequence into a more manageable form.

Algorithm 1 Pseudocode for Simplification of Opcode Sequences

Input: Opcode sequence
Output: Simplified opcode sequence

```
simplified_opcode_sequence = [ ]                ▷ Initialize as an empty list
for each opcode in the opcode sequence do
    if opcode belongs to a predefined category then
        Add the category label to simplified_opcode_sequence
    else
        Add the original opcode to simplified_opcode_sequence
    end if
end for
return simplified_opcode_sequence
```

To provide an example of how an opcode sequence is represented before and after simplification, we present a sample sequence and demonstrate how it appears with the original opcodes versus their corresponding high-level categories in Table 4.3.

This categorization helps in simplifying the analysis by abstracting individual opcodes into broader groups, making it easier to identify patterns and conduct further

[6] https://github.com/crytic/pyevmasm

research on the behaviour and structure of smart contracts.

After simplifying the opcode sequences as described in Algorithm 1, we proceed to generate embeddings for these sequences to transform them into a format suitable [110] for machine learning models [111].

To achieve this, we use a learned word embedding approach to convert the simplified opcode sequences into dense vectors. We chose this approach due to its ability to adapt and capture nuanced relationships within the data [110, 112]. Unlike static embeddings, learned embeddings adjust based on the specific dataset, optimizing the representations for our analysis of smart contract reputability. This adaptability is crucial for capturing the context-specific patterns in cryptocurrency and financial data [113, 114].

To generate these embeddings for opcode sequences, we first convert each opcode sequence into integer representations using the Tokenizer from TensorFlow's Keras API [115] which assigns a unique integer to each distinct opcode. After tokenization, the sequences are padded to ensure uniform length across all sequences. This padding is crucial for maintaining consistent input dimensions, which are required for neural network models.

Next, the integer-encoded opcodes are processed through an Embedding layer in a TensorFlow model. This layer transforms each integer into a dense vector representation, where each opcode is represented by a 50-dimensional vector.

To obtain a single embedding vector for each smart contract, we calculate the mean of the embeddings for all opcodes in the sequence. This aggregated vector represents the entire opcode sequence for each smart contract. This process converts the high-dimensional and categorical opcode sequences into a dense vector space, facilitating the application of various machine learning techniques to predict and analyse smart contract reputability.

Moreover, we standardize the opcode sequences by scaling the feature values so that each sequence has a consistent range. This process involves transforming the data such that numerical features have a mean of 0 and a standard deviation of 1, ensuring that all input sequences contribute equally to the model training. This step helps mitigate the effects of varying scales among sequences and promotes more stable and effective learning by the machine learning algorithms.

4.3 Handling a Skewed Class Distribution

In this section, we address the issue of class imbalance in the dataset, particularly the under-representation of illicit smart contracts. This effort aligns with Objective 1 (Section 1.3), which aims to acquire, preprocess, and analyse a balanced dataset of smart

contracts to evaluate the performance of boosting algorithms in predicting reputability based on bytecode embeddings. To mitigate the class imbalance, we apply a combination of data augmentation techniques—namely, SMOTE, ADASYN, and GANs—and compare their effectiveness in improving model performance, with particular emphasis on recall and F1-Score as key metrics.

SMOTE and ADASYN generate synthetic samples by interpolating in the feature space of high-dimensional embeddings of smart contract bytecode [116]. These embeddings, learned via models like Word2Vec, represent code structure and context but not actual opcodes [117]. While interpolated vectors may not correspond to valid code, they are used solely to augment training data, based on the assumption that the embedding space preserves meaningful semantic relationships. Though approximate, this method is common in fields like NLP and malware detection [118] and showed performance gains—though GAN-based augmentation proved more effective.

4.3.1 Evaluation of SMOTE and ADASYN

Both SMOTE and ADASYN were applied to augment the minority class of illicit smart contracts. As discussed in Section 2.6, SMOTE generates synthetic data by interpolating between minority class samples, while ADASYN adapts the number of synthetic samples generated based on the difficulty of the sample. After applying each technique, the augmented datasets are used to train the anomaly detection models, and their performances are compared using Recall and F1-score (Section 2.5).

We also evaluate the model's performance using the original dataset, the SMOTE-augmented dataset, and the ADASYN-augmented dataset to observe how these augmentation techniques impact the model's ability to detect illicit smart contracts.

4.3.2 Comparison with GAN-Based Augmentation

While SMOTE and ADASYN generate synthetic data by interpolating between existing data points, GANs offer a different approach by producing more diverse and realistic synthetic samples. Since GANs are designed to generate synthetic illicit smart contract transactions that enhance the balance of the class distribution, capturing more complex patterns and potentially leading to higher-quality synthetic data, we expect this capability to create more representative samples of the minority class.

To evaluate the quality of the synthetic data generated by GANs, several statistical measures are employed. The Mean Absolute Deviation (MAD) quantifies the average absolute difference between the real and synthetic samples, providing an indication of how closely the synthetic data resembles the real data. The equation for MAD is:

$$\text{MAD} = \frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{x}_i| \tag{4.1}$$

where x_i represents the real sample and $\hat{x}_i$ denotes the synthetic sample.

The Variance Ratio [119] compares the variance of the synthetic dataset with that of the real dataset. This metric helps in understanding whether the variability of the synthetic data aligns with the real data:

$$\text{Variance Ratio} = \frac{\text{Var}_{\text{synthetic}}}{\text{Var}_{\text{real}}} \tag{4.2}$$

where $\text{Var}_{\text{synthetic}}$ and Var_{real} are the variances of the synthetic and real datasets.

The Correlation Coefficient [120] quantifies the strength and direction of the linear relationship between two datasets. It takes a value between -1 and $+1$, where $+1$ indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 suggests no linear relationship. It is defined as:

$$\text{Correlation Coefficient} = \frac{\text{Cov}(X, \hat{X})}{\sigma_X \sigma_{\hat{X}}} \tag{4.3}$$

where $\text{Cov}(X, \hat{X})$ is the covariance between real and synthetic samples, and σ_X and $\sigma_{\hat{X}}$ are the standard deviations of the real and synthetic samples, respectively.

Additionally, the Fréchet Inception Distance (FID) [121] score is used to measure the quality of the generated samples by comparing the distribution of real and synthetic data in the feature space of a pre-trained Inception model. A lower FID score indicates that the synthetic data is more similar to the real data:

$$\text{FID} = \|\mu_r - \mu_g\|^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}) \tag{4.4}$$

where μ_r and Σ_r are the mean and covariance of the real data features, and μ_g and Σ_g are those of the synthetic data features.

These metrics collectively provide a comprehensive evaluation of the GAN-generated data's quality, highlighting how well it mimics the real data distribution and whether it effectively enhances the balance of the class distribution.

To optimize the GAN's performance, hyperparameters were tuned within the ranges shown in Table 4.4.

The choice of these hyperparameters was guided by the need to balance training stability, model capacity, and generalization ability, ensuring that the GAN could generate high-quality synthetic samples while effectively learning from the data. To identify the optimal set of hyperparameters, we employed a random search strategy, which allowed us to explore a broad range of hyperparameter combinations efficiently. This method helped find configurations that maximize model performance, as highlighted in

Hyperparameter	Search Space
Learning Rate for Generator (lr_g)	{0.0001, 0.0002, 0.0005}
Learning Rate for Discriminator (lr_d)	{0.0001, 0.0002, 0.0005}
Batch Size	{32, 64, 128}
Input Dimension	{50, 100, 200}
Dropout Rate	{0.2, 0.3, 0.4}
Hidden Units	{128, 256, 512}

Table 4.4 Search space for GAN hyperparameters.

recent literature on GAN hyperparameter tuning [122, 123].

4.3.3 t-SNE and PCA Visualization and Dataset Comparison

We further evaluate the similarity between the real and synthetic datasets by visualizing the two datasets using t-distributed Stochastic Neighbor Embedding (t-SNE)) [124] and Principal Component Analysis (PCA). This allows us to observe how well the synthetic data generated by the GAN approximates the distribution of the real data. By projecting both datasets into a lower-dimensional space, we can qualitatively assess how closely the synthetic samples resemble the real samples, ensuring that the GAN does not introduce unrealistic artifacts into the data.

The models trained on the original dataset, the SMOTE-augmented dataset, the ADASYN-augmented dataset, and the GAN-generated dataset are compared in terms of recall and F1-Score. Given that the detection of illicit smart contracts is our primary concern, we expect GAN-based augmentation to outperform SMOTE and ADASYN, as it is capable of generating higher-quality and more diverse samples

4.4 Experiment 1: Embedding-based Bytecode Analysis

This section contributes to the latter part of Objective 1 (Section 1.3), which involves analysing the dataset and evaluating boosting algorithms. We utilize features derived from smart contract bytecode to develop predictive models aimed at assessing reputability. While prior studies often rely on traditional static analysis to identify syntactic or semantic vulnerabilities from source code or intermediate representations [15], our approach differs. We do not perform traditional static analysis; instead, we apply AI techniques directly to the compiled bytecode of labelled smart contracts (reputable vs. illicit) thus performing an embedding-based bytecode analysis.

Specifically, we generate high-dimensional code embeddings from bytecode and use these as input to machine learning models. This allows the models to learn behavioural patterns indicative of illicit activity, independent of source-level variability.

Moreover, focusing on bytecode better reflects the actual deployed form of contracts on the blockchain, enhancing the relevance of the analysis.

The simplified opcode sequences, which have been categorized into high-level operation groups defined in Table C.1, serve as the foundational features for our analysis. By converting these sequences into dense vector representations through a learned embedding approach, we transform the high-dimensional, categorical opcode data into a format suitable for machine learning models. This process not only reduces the complexity of the data but also enables the model to capture nuanced patterns that are critical for predicting reputability.

The dataset, consisting of opcode embeddings and vulnerability features, was first augmented using various resampling techniques to address the class imbalance between reputable and illicit smart contracts. These techniques, including SMOTE, ADASYN, and GANs-based augmentation, were applied as detailed in Section 4.3. This step was critical for ensuring that the models were trained on a balanced dataset, thereby improving their ability to generalize and accurately predict reputability across different types of smart contracts.

Following the augmentation, the enriched dataset was used to train and compare three different gradient boosting models: XGBoost, CatBoost, and LightGBM. These models were selected for their proven effectiveness in handling large, high-dimensional datasets, which aligns well with the nature of our data. Gradient boosting techniques are recognized for their efficiency, adaptability, and capability to manage complex data patterns, especially in financial and cryptocurrency applications [44, 46, 125]. Each model has its own set of strengths, making them suitable candidates for our analysis of smart contract reputability.

To determine the most effective model, the opcode embeddings and vulnerability features were used as input to each of these models. The original dataset was divided into majority (reputable, `is_reputable = 1`) and minority (illicit, `is_reputable = 0`) classes. The majority class data was further split into training and testing sets, with 70% allocated for training. This approach is commonly used in machine learning to ensure that there is enough data for both training and evaluation, allowing for robust model generalization. The synthetic minority class data was then combined with the majority class training data, using techniques such as SMOTE to address class imbalance, as is standard in imbalanced classification tasks [52, 126]. For evaluation, we used the original majority class test data and real minority class data to assess the models' performance on the authentic dataset while training on the synthetic data. This evaluation approach is consistent with similar studies, which utilize separate training and testing datasets to avoid overfitting and ensure that performance is assessed on unseen, real-world data [127].

A comprehensive hyperparameter tuning strategy using GridSearchCV across 5

folds was employed to optimize model performance across the hyperparameters shown in Table 4.5. The parameter ranges were selected based on standard practices in the literature and prior experience with gradient boosting models. The chosen values for maximum depth (2–8) and number of estimators (100–300) ensure a balance between underfitting and overfitting, while the learning rate values (0.2, 0.1, 0.01) allow for fine-tuning the model's ability to generalize. These ranges are typical for gradient boosting models and have been used in similar studies [128, 129].

Furthermore, GridSearchCV was used which involves evaluating different hyperparameter combinations by training the model on various subsets of the data, providing robust validation and preventing overfitting. In each iteration, four subsets were used for training, while the remaining subset served as the validation set. This process was repeated for all folds to ensure that every subset had the opportunity to be a validation set. The hyperparameters tuned included learning rates, the number of boosting rounds, maximum tree depth, and regularization parameters, among others.

Hyperparameter	**Search Space**
Maximum Depth (max_depth)	{2, 3, 4, 5, 6, 7, 8}
Number of Estimators (n_estimators)	{100, 150, 200, 250, 300}
Learning Rate (learning_rate)	{0.2, 0.1, 0.01}

Table 4.5 Search space for gradient boosting machine hyperparameters.

Following hyperparameter tuning, a thorough evaluation was conducted to compare model performance in predicting the reputability of smart contracts. The evaluation metrics included accuracy, precision, recall, and F1-score, which were used to assess each model's ability to distinguish between reputable and illicit contracts. This comparison not only identified the most effective model for our dataset but also provided insights into how each gradient boosting technique handles the complexities of smart contract data, particularly regarding class imbalance and high-dimensional feature spaces.

4.4.1 Evaluating Embedding-based Bytecode Analysis

To evaluate the effectiveness of the embedding-based bytecode analysis models, we employed key classification metrics, including accuracy, precision, recall, and F1-score, as detailed in Section 2.5. These metrics offer a thorough assessment of the models' ability to distinguish between reputable (positive class) and illicit (negative class) smart contracts.

The core concepts necessary for calculating these metrics, such as true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN), are defined in Section 2.5. Using these definitions, the metrics were calculated following the standard

formulas provided.

These evaluation metrics were applied to each model after the hyperparameter tuning process to determine the best-performing model in terms of accurately predicting the reputability of smart contracts.

4.5 Experiment 2: Temporal Sequence Analysis

Once the bytecode-level analysis was conducted, we transitioned to temporal analysis to examine how reputability evolves over time. Our focus then shifted to the characteristics of the transactions themselves. Consistent with the findings of [11] and [8], we concentrated primarily on normal and internal transactions since they were identified as particularly important for providing meaningful insights into the behaviour and reputability of the contracts.

In order to capture both temporal dependencies and local patterns within transaction sequences, we implemented two types of autoencoders: LSTM-based and CNN-based autoencoders. These models are well-suited for detecting anomalies in blockchain transaction data by identifying deviations from learned transaction patterns. Our methodology for anomaly detection relies on the reconstruction error—differences (Equation. 2.14) between the original input sequences and their reconstructions by the autoencoder models—which has been widely adopted in similar anomaly detection frameworks within blockchain analytics [130–132].

4.5.1 Feature Extraction and Model Training

In preparing the dataset for temporal analysis, it was crucial to organize the data in a format suitable for time series analysis. Temporal analysis of blockchain data often requires a structured approach where transactions are chronologically arranged to reveal patterns and trends over time. Studies by [133] and [60] highlight the importance of time-ordered data in financial and blockchain transaction analysis, underscoring the necessity for careful temporal organization.

In this study, outlier removal was performed to ensure that extreme data points did not unduly influence model performance. The dataset originally contained 1,321,035 transactions, of which 493,354 were identified as outliers using the 1.5 * IQR rule, resulting in the removal of approximately 37.4% of the data. This method, though aggressive, is a standard approach in anomaly detection tasks where the goal is to focus on typical transaction behaviours and remove noise that could distort learning [134]. Outliers in the context of smart contracts, such as extreme values in transaction counts, gas usage, or error rates, are often indicative of irregularities or data errors, which are not helpful for training robust models [132]. By applying this outlier removal process, we ensured

that the model could focus on the more relevant and typical patterns in the data, improving its ability to generalize and detect meaningful anomalies in smart contract behaviour.

Features were initially extracted at an hourly granularity to capture short-term fluctuations crucial for anomaly detection. While daily aggregation could be advantageous for simplifying analysis, as it highlights broad trends and helps identify long-term shifts in transaction behaviour, it comes with drawbacks. This level of aggregation may overlook critical short-lived spikes and sudden deviations, which are vital for detecting illicit activities and subtle anomalies that occur over shorter timeframes.

While minute-level granularity could provide a more detailed view of transaction behaviours and potentially capture rapid anomalous events, it proved computationally impractical due to the sheer volume of data and increased training complexity. Processing data at this level would require significant computational resources and extended training times, without a clear improvement in model performance.

Therefore, we selected hourly aggregation as the optimal balance, capturing short-term variations like transaction bursts and longer-term trends, while maintaining computational feasibility. This granularity provided enough temporal detail to detect anomalies without the over-smoothing issues of daily aggregation.

The features collected include aggregated transaction properties such as total and mean value, gas usage, gas price, and transaction error rates, as these metrics are strongly correlated with anomalous activity in blockchain transactions [8, 17]. These features provide critical insights into transaction behaviour, aiding in the detection of unusual patterns that may indicate potential issues. The complete list of 32 features is provided in Appendix A.

After preprocessing the transaction data and normalizing it using a MinMaxScaler, sequences of transaction features are grouped by smart contract address and used as input for both the LSTM-based and CNN-based autoencoder models. As discussed in Section 2.8.2, the LSTM autoencoder is well-suited for capturing long-term temporal dependencies, as LSTMs are designed to maintain information across sequential data, allowing the model to detect anomalies arising from patterns developing over time [79]. In blockchain transactions, where irregularities can manifest across consecutive hours, the LSTM autoencoder's ability to model temporal dependencies is essential.

On the other hand, the CNN autoencoder excels at detecting localized patterns within the data. By applying convolutional filters, the CNN model is adept at identifying sharp deviations or sudden changes in the transaction sequences, such as short bursts in transaction frequency or spikes in gas usage, which are indicative of illicit activities in blockchain transactions [80]. The CNN's ability to capture such local variations is particularly valuable in contexts where anomalies may manifest over shorter periods.

For both models, we used a maximum sequence length of 24 timesteps, corresponding to 24 hours of transaction data. This decision ensures fairness in the evalu-

ation of both models, as they are provided with the same temporal context—one day of transactional behaviour. This uniform sequence length ensures that both the LSTM and CNN models are evaluated under the same conditions, allowing us to compare their performance more effectively.

By using this consistent sequence length, the LSTM model is able to capture longer-term trends and dependencies within a 24-hour window, while the CNN model focuses on identifying short-term irregularities within the same timeframe. This approach allows us to leverage both models' strengths, providing a comprehensive evaluation of anomalies across different timescales in the blockchain transactions.

To ensure optimal performance of both the convolutional autoencoder (CAE) and LSTM-based autoencoder models, we conducted a thorough hyperparameter search using Keras Tuner [135]. The hyperparameters considered during tuning included the number of filters, kernel size, activation function, dropout rate, L2 regularization, and learning rate for the CAE and LSTM-based autoencoders. For example, dropout was included as a form of regularization to prevent overfitting, particularly in scenarios with limited anomalous events [136] and L2 regularization was used to penalize complex models, improving generalization [137].

The search space for hyperparameters was carefully designed to strike a balance between model complexity and computational efficiency. For instance, larger kernel sizes and filter numbers could potentially capture more complex patterns, but they also significantly increase the computational load. The decision to test kernel sizes of 3 and 5 was guided by prior studies indicating that smaller kernel sizes are effective for identifying local patterns in time series data, particularly in blockchain contexts [138].

Hyperparameter tuning was conducted using `TimeSeriesSplit` cross-validation, which is particularly suitable for time series data as it preserves the temporal order by ensuring that training data always precedes validation data. This prevents data leakage from future observations into the training process, a critical consideration for modelling real-world scenarios where the goal is to predict future outcomes based on past behaviour [139]. We used five splits (`n_splits=5`), resulting in five sequential train-validation pairs. In each fold, earlier data was used for training and the immediately following segment for validation. Each fold gradually increased the training size and shifted the validation window forward, enabling robust evaluation across different time intervals. The final model was evaluated on a held-out test set consisting of the latest segment of the time series data that was not used in any training or validation folds, ensuring a realistic forward-testing scenario.

The final hyperparameter tuning results were averaged across the cross-validation folds to ensure robust model selection. This approach allowed us to derive the best-performing configuration for each model type, with a focus on minimizing reconstruction error for effective anomaly detection in blockchain transactions. The hyperparam-

Hyperparameter	CAE Search Space	LSTM Search Space
Filters (1st layer)	16 to 64 (step 16)	-
Kernel Size (1st layer)	3, 5	-
Activation Function	relu, tanh	relu, tanh
L2 Regularization (1st layer)	0 to 0.1 (step 0.01)	0 to 0.1 (step 0.01)
Pool Size (1st layer)	2, 4	-
Dropout Rate (1st layer)	0 to 0.5 (step 0.1)	0 to 0.5 (step 0.1)
Filters (2nd layer)	8 to 32 (step 8)	-
Kernel Size (2nd layer)	3, 5	-
L2 Regularization (2nd layer)	0 to 0.1 (step 0.01)	0 to 0.1 (step 0.01)
Pool Size (2nd layer)	2, 4	-
Up Sampling Size (Decoder)	2, 4	-
LSTM Units (1st layer)	-	16 to 128 (step 16)
LSTM Units (2nd layer)	-	8 to 64 (step 8)
Batch Size	16, 32, 64	16, 32, 64
Learning Rate	10^{-4} to 10^{-2} (log scale)	10^{-4} to 10^{-2} (log scale)

Table 4.6 Hyperparameter search space for CAE and LSTM Autoencoder models

eter tuning process was instrumental in finding the balance between capturing intricate transaction patterns and maintaining generalizability across different addresses.

The inclusion of multiple values for each hyperparameter enabled us to systematically explore a wide range of model configurations to determine the optimal settings for anomaly detection in blockchain transactions. This systematic exploration not only optimized model performance but also provided insights into how different hyperparameters influence the ability to detect anomalous behaviour in transaction sequences.

4.5.2 Anomaly Detection Using Reconstruction Error

The primary method for detecting anomalies in smart contract behaviour is based on the reconstruction error of the LSTM and CNN autoencoders. After training both models on normal transaction sequences (i.e., transactions conducted by reputable smart contracts), they are tested on unseen sequences. The reconstruction error for each sequence is computed as the Mean Absolute Error (MAE) between the original input and its reconstruction. High reconstruction errors indicate sequences that deviate from the learned normal behaviour, signalling potential anomalies.

The **MAE** is calculated as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{x}_i| \tag{4.5}$$

where:

n is the number of elements in the sequence
x_i is the original input at time step i
$\hat{x}_i$ is the reconstructed value at time step i

To ensure a fair evaluation and avoid data leakage, the dataset was split at the contract address level rather than at the transaction level. Reputable addresses were initially divided into a 70/30 split, with 70% used for training the autoencoders and 30% reserved for validation and testing. The 30% set was then further divided into 80% for reputable validation (232 addresses) and 20% for reputable test evaluation (59 addresses). Similarly, illicit contract addresses were split into validation (53 addresses) and test (23 addresses) sets. Threshold tuning for anomaly detection was performed using both the reputable and illicit validation sets by analysing their reconstruction error distributions. The final threshold was selected based on maximizing the separation between the two classes. The test set—comprising transactions from distinct, previously unseen reputable and illicit addresses—was strictly held out during training and validation, and only used during the final evaluation phase to assess generalization and anomaly detection accuracy.

The choice of reconstruction error as the primary anomaly indicator is driven by its proven ability to detect deviations from learned normal behaviour, a method that is widely used in unsupervised anomaly detection tasks, particularly in high-dimensional and complex domains like blockchain transactions [140]. In these settings, where normal behaviours are intricate and often non-linear, reconstruction error provides an effective and intuitive means of identifying unusual patterns that could signal illicit activity. By training both the LSTM and CNN autoencoders on normal transaction sequences, the reconstruction error highlights sequences that deviate from the established baseline, making it ideal for anomaly detection in blockchain systems [14, 84].

Alternative approaches, such as direct classification models that predict whether a sequence is anomalous based on predefined features, could also be considered. However, these models often require large amounts of labelled data, which is scarce in anomaly detection tasks, and they are prone to overfitting, especially in highly imbalanced settings where anomalies are rare. In contrast, reconstruction-based methods are inherently unsupervised, making them more flexible in environments like blockchain systems, where anomaly examples are infrequent and varied [14].

Moreover, reconstruction error is favored for its interpretability—higher errors directly correlate with more significant deviations from normal behaviour, which is particularly valuable when investigating potential anomalies. The magnitude of the recon-

struction error provides an indication of the severity of the anomaly, making it easier to assess the importance of the detected deviation. This interpretability, combined with the robustness and flexibility of unsupervised learning, makes reconstruction error an ideal choice for detecting anomalies in blockchain transactions, where the patterns of illicit behaviour can be unpredictable and irregular [83, 85].

To evaluate the anomaly detection performance, we calculate the reconstruction error for each transaction and then aggregate these errors at the contract level, as our labels are defined at the contract level. Specifically, we take the mean reconstruction error for each contract and determine the optimal threshold using percentile-based thresholding techniques (as outlined in Algorithm 2). This approach allows us to balance detecting the most anomalies while maximizing the F1-score for illicit smart contracts.

We selected a search space for the transaction-level thresholds between the 75th and 90th percentiles. This range was chosen based on prior research in anomaly detection within financial systems, where percentile-based thresholds have been successfully employed to detect outliers or illicit behaviour. In this context, the 75th percentile often captures a sufficient number of anomalies without overfitting, while the 90th percentile focuses on the most extreme cases. By experimenting within this range, we aim to find a balance between precision and recall, which is particularly critical when detecting anomalies in blockchain transactions, as highlighted by studies on detecting fraudulent or abnormal behaviour in blockchain-based financial systems.

The percentile-based thresholding is a widely used technique in anomaly detection, particularly in high-dimensional, time-series data, as seen in cryptocurrency fraud detection frameworks [141–143]. We adopt the same thresholding technique to maximize detection performance in blockchain environments, where the variability and distribution of normal and illicit behaviours can differ significantly.

Thus, our chosen thresholds reflect best practices in financial anomaly detection and are specifically tuned for detecting complex, illicit behaviour in blockchain transactions.

4.5.3 Multimodal Data Fusion

After training both the LSTM and CNN autoencoders independently, we extend our model by incorporating opcode embeddings as additional features for each transaction. These opcode embeddings provide crucial code information, adding context to each transaction beyond purely transactional data.

In this phase, we prioritize the model that demonstrates the best performance on transactional data alone. Based on its superior results in terms of anomaly detection (namely the F1-score for illicit contracts), we apply the same model to test multimodal data fusion. This fusion integrates both the transactional data and opcode embeddings,

Algorithm 2 Threshold Optimization for Anomaly Detection

Input:
Reconstruction errors $R = \{r_1, r_2, \ldots, r_n\}$
Contract status labels $L = \{l_1, l_2, \ldots, l_m\}$
Output:
Optimal threshold values T^* and C^* for anomaly detection
$T \leftarrow \{75, 80, 85, 90\}$ (percentiles for transaction-level threshold)
$C \leftarrow \{0.1, 0.2, \ldots, 0.5\}$ (contract-level anomaly thresholds)
$F_1 \leftarrow 0$
$P \leftarrow \emptyset$ (Best parameters)
for $t \in T$ **do**
$\theta_t \leftarrow$ value at $\frac{t}{100} \times (n-1)$ in sorted R
$\forall i,\ r_i > \theta_t \implies$ anomaly indicator$_i = 1$
$\forall i,\ r_i \leq \theta_t \implies$ anomaly indicator$_i = 0$
$a_c \leftarrow \frac{1}{N_c} \sum_{i=1}^{N_c} \mathbf{1}(r_i > \theta_t)$ for each contract c
where N_c is the number of transactions associated with contract c.
Predicted illicit$_c \leftarrow \mathbf{1}(a_c > c)$ for each contract c
$F_1' \leftarrow \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
where
Precision $= \frac{\text{TP}}{\text{TP+FP}}$, Recall $= \frac{\text{TP}}{\text{TP+FN}}$
TP, FP, and FN are calculated based on merging predicted and actual labels from L
if $F_1' > F_1$ **then**
$F_1 \leftarrow F_1'$
$P \leftarrow \{t, c\}$
end if
end for
return F_1, P

enhancing the model's capability to detect more subtle and complex anomalies. By leveraging both dynamic transaction behaviour and bytecode embeddings, the fused model can capture a wider range of anomalies that may not be detectable using transaction data or opcode embeddings alone.

This method aligns with the principle that combining complementary data sources can significantly boost anomaly detection performance, as seen in previous research [87, 144]. For instance, while the chosen autoencoder model excels at detecting temporal or spatial patterns within transaction data, the added opcode embeddings improve its sensitivity to anomalies tied to the underlying contract code, providing a more holistic view of illicit smart contract behaviour.

4.5.4 Evaluation of Anomaly Detection Models

The performance of both the LSTM and CNN autoencoders, as well as the multimodal fusion model, is evaluated using standard metrics for anomaly detection. The models' loss functions are defined using the MSE, which is widely used in autoencoder-based anomaly detection tasks [145]. MSE is chosen because it effectively captures the magnitude of the reconstruction error, providing a clear signal when the model fails to accurately reconstruct an input, which is critical for detecting anomalous patterns in smart contract transactions.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 \tag{4.6}$$

where:

n is the number of elements in the sequence,
x_i is the original input at time step i,
$\hat{x}_i$ is the reconstructed value at time step i.

Higher MSE values indicate poor reconstruction and, consequently, a higher likelihood of an anomaly. After calculating the reconstruction error for each transaction, an empirically determined threshold is applied to classify transactions as anomalous.

At the contract level, we aggregate transaction-level anomaly predictions to make a final decision. A smart contract is classified as anomalous if more than 30% of its transactions are flagged as anomalous. This 30% threshold was not arbitrarily selected but determined through empirical tuning on the validation set [146]. We performed a grid search over a range of transaction-level reconstruction error percentiles and contract-level anomaly thresholds (ranging from 10% to 70%), evaluating each combination using the F1-score. The 30% threshold yielded the best performance in distinguishing be-

tween reputable and illicit contracts. This strategy ensures that the model balances sensitivity and specificity at the contract level while aligning with prior anomaly detection approaches that apply heuristic or data-driven aggregation rules.

We apply this approach to ensure that contracts with a high percentage of anomalous transactions are identified as potentially fraudulent, in line with methodologies used in blockchain fraud detection [142]. The models are then evaluated in a similar fashion as embedding-based bytecode analysis evaluation using the recall, accuracy etc. as defined in Section 4.4.1.

Moreover, we also visualize the distribution of reconstruction errors, as well as the Receiver-operating characteristic curve (ROC) curve and the impact of threshold adjustments on true positive and false positive rates. These visualizations allow us to fine-tune the threshold for anomaly detection and ensure that the model effectively captures both gradual and sudden changes in transaction behaviour.

Overall, the multimodal data fusion model is expected to outperform the individual LSTM and CNN autoencoder models in terms of anomaly detection accuracy, as it leverages both temporal and local spatial features. By applying this methodology, we gain deeper insights into the evolving reputability of smart contracts and improve our ability to detect illicit activities within blockchain ecosystems.

4.6 Experiment 3: Trend Analysis for Reputability Scores

Following the anomaly detection in Experiment 2 (Section 4.5), which assigned reconstruction errors to each smart contract over time, these errors were transformed into reputability scores ranging from 0 to 1. Experiment 3 addresses Objective 3 (Section 1.3), focusing on evaluating long-term trends in these scores. This is essential as the reputability scores from hourly transactional data in Experiment 2 are prone to significant noise. While anomaly detection provides detailed insights into contract behaviour, this noise can obscure important patterns for long-term investors highlighting broader reputability trends over momentary fluctuations. This experiment uses sequence modelling to analyse how reputability evolves over time, offering more actionable insights.

4.6.1 Data Preprocessing

To align with the embedding-based bytecode analysis scores from Section 4.4, the reputability score was obtained by converting the reconstruction error through the following process. First, the reconstruction error was normalized between 0 and 1. The normalized score was then passed through the sigmoid function:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{4.7}$$

The sigmoid function was chosen for its property of mapping real-valued inputs into a bounded range of $(0, 1)$, making the interpretation as a probability-like score intuitive [147]. Specifically, in our context, the sigmoid provides a smooth transition that compresses high reconstruction errors (which are indicative of more anomalous, less reputable behaviour) towards values closer to 0, while mapping lower reconstruction errors to values closer to 1.

The sigmoid's non-linearity is useful in capturing the different levels of severity in reconstruction errors [40]. High errors, which indicate severe anomalies, are strongly pushed towards 0, emphasizing their lower reputability, whereas smaller errors—suggestive of normal behaviour—are pushed closer to 1. Thus, the sigmoid transformation ensures a direct, interpretable mapping from reconstruction error to reputability that is consistent with the notion of reliability [148].

After passing through the sigmoid function, a complement of this output was taken, where the final reputability score r was defined as:

$$r = 1 - S(x) \tag{4.8}$$

This final transformation was made to ensure that illicit contracts are assigned scores closer to 0, whereas reputable contracts receive scores closer to 1, aligning the reputability metric across different datasets and experiments.This approach is consistent with other areas of this research, where higher values indicate higher reputability, thereby allowing for a unified and coherent interpretation of reputability scores throughout the entire analysis and facilitating comparisons across different stages of the study.

The reputability scores computed were based on hourly aggregated data. Since the transactional volume of illicit contracts is typically lower than reputable ones, using a large window size (e.g., 50) for temporal analysis would eliminate many contracts that do not meet the minimum required transaction count. Hence, a window size of 25 was chosen to provide a balance between capturing sufficient temporal patterns and retaining as many contracts as possible for analysis. The hourly aggregation of transactional data reflects a compromise between computational feasibility and capturing temporal variation, with references indicating that shorter windows may fail to capture meaningful temporal dependencies, while longer windows may lead to excessive loss of data due to sparsity [149, 150].

To avoid data leakage and ensure consistent scaling across datasets, the normalization of reconstruction errors was performed using the minimum and maximum values computed from the training set only. These min-max values were then used to normalize both training and test reconstruction errors into the $[0, 1]$ range. Following normalization,

the sigmoid function was applied uniformly to both sets. This ensured that the reputability scores in the test set were scaled and interpreted consistently with the training set.

4.6.2 Model Development and Hyperparameter Tuning

To analyse long-term trends in the reputability scores of smart contracts, two predictive models were developed: an LSTM model and a CNN-LSTM model with Multi-Head Attention. These models were used to identify the temporal evolution of reputability, focusing on capturing both long-term dependencies and critical points within transactional sequences.

The **LSTM model** was constructed to capture temporal dependencies using sequential patterns of reputability scores. An embedding layer was used to incorporate smart contract information by converting categorical smart contract addresses into dense numerical representations. This enriched the model by capturing similarities between contracts based on their historical behaviour. The LSTM layer extracted temporal patterns from the time series data, followed by concatenation with the contract embeddings, and a dense layer was used for reputability score prediction.

The **CNN-LSTM model with Multi-Head Attention** was designed to enhance the model's ability to capture both local and long-term features in the data. A CNN layer was used to extract local, short-term features from the reputability sequences, while an LSTM layer captured temporal dependencies. The Multi-Head Attention mechanism allowed the model to focus on key points in the sequence, adaptively emphasizing the most informative transactions. Like the LSTM model, this architecture also included a contract embedding layer to enrich the model's understanding of different contracts, followed by a concatenation layer and a final dense layer for score prediction.

For both models, the dataset was split at the contract level into training, validation, and test sets (70%, 15%, 15%), ensuring that all transactions for a given contract were assigned to the same split to prevent data leakage. This allowed the models to learn meaningful patterns without introducing bias between training and validation or test data. The features were also scaled using the MinMaxScaler, which normalized the feature values to ensure they were on the same scale, improving model training stability.

Furthermore, contract embeddings (Section 2.8) were trained jointly with the rest of the model, allowing the model to learn an optimal representation of each smart contract for the prediction task. This helped to reduce overfitting, as the model learns a compressed, representative form of each contract, which generalizes better when provided with unseen data.

Hyperparameter Tuning

To achieve optimal performance, hyperparameters for both models were tuned using K-Fold Cross Validation ($K = 3$) to balance computational feasibility with evaluation robustness. Given the data's temporal nature and overfitting risk, this step was essential. The tuned hyperparameters included LSTM units, batch size, learning rate, and other model-specific parameters, as detailed in Table 4.7. This process ensured robustness across dataset subsets, providing a reliable estimate of generalizability.

The search space for hyperparameters used during the tuning process is summarized in Table 4.7. The window size was fixed at 25, as this provided an optimal balance between sequence length and data retention. A window size of 50 was also experimented with, but many illicit contracts were excluded due to the limited number of transactions. By reducing the window size to 25, we retained a higher number of sequences for training, especially from illicit contracts.

Hyperparameter	Search Space
LSTM Model	
Window Size	{25, 50}
LSTM Units	{32, 64, 128}
Batch Size	{32, 64}
Learning Rate	{0.001, 0.0005, 0.0001}
CNN-LSTM with Multi-Head Attention Model	
Number of Filters	{32, 64, 128}
LSTM Units	{32, 64, 128}
Learning Rate	{0.001, 0.0005, 0.0001}
Activation	{relu, tanh}
Dropout Rate	{0.1, 0.2, 0.3}
L2 Regularization	{1e-4, 1e-3, 1e-2}

Table 4.7 Search space for LSTM and Multi-Head Attention CNN-LSTM hyperparameters.

4.6.3 Trend Analysis Approach and Evaluation Strategy

In Experiment 3, the trend analysis approach focused on capturing the long-term evolution of reputability scores in smart contracts, aiming to identify gradual changes that could signal emerging risks or sustained reputability. Unlike short-term anomaly detection, this experiment emphasized modelling continuous reputability trends across various contract sequences, providing a nuanced view of how reputability dynamics develop over time without being obscured by momentary fluctuations.

The evaluation strategy for Experiment 3 comprised the following elements:

- **Quantitative Evaluation - Reputability Score Consistency and Stability Analysis**: We evaluated the model's ability to predict reputability scores over time, focusing on consistency and stability across different contracts. Key metrics included Mean Absolute Error (MAE) and Mean Squared Error (MSE) between predicted and actual reputability scores, which helped quantify the model's accuracy in approximating real-world reputability. Lower error values indicated better model alignment with actual trends, underscoring its capability to capture evolving reputability patterns accurately.

- **Quantitative Evaluation** - To assess the temporal alignment between predicted and actual reputability scores, we employed cross-correlation analysis. By examining correlations across varying time lags, we evaluated the model's ability to produce synchronized predictions, effectively capturing reputability trends that unfold over time. This method validated the model's responsiveness to shifts in reputability within the sequence of contract interactions.

- **Cross-Correlation Analysis** - Complementing the quantitative evaluation, we visually inspected the predicted and actual reputability trends for selected smart contracts. By plotting the evolution of reputability scores, we aimed to identify alignment and deviations between predicted trends and observed reputability trajectories. This visual component provided intuitive insights into the model's reliability, allowing us to assess its practical utility in capturing long-term reputability shifts accurately.

The models were evaluated on the test set, focusing on their effectiveness in predicting these trends compared to the short-term anomaly detection used in Experiment 2. The **LSTM model** served as a baseline due to its known effectiveness in capturing temporal dependencies, whereas the **CNN-LSTM with Multi-Head Attention** was expected to enhance performance by selectively attending to key parts of the transaction sequence. The use of these two models aimed to determine whether the more advanced architecture could improve the reliability of long-term trend prediction.

This approach ensured that the evaluation not only covered the accuracy of the model but also provided a deeper understanding of its practical applicability through visual and intuitive means.

4.7 Summary

The methodology implemented in this study systematically addresses the defined objectives through three core experiments, each focusing on a distinct aspect of smart contract reputability analysis.

Experiment 1: Boosted Embedding-based Bytecode Analysis

Aligned with Objective 1, this experiment focuses on acquiring, preprocessing, and analysing a balanced dataset of smart contracts to evaluate boosting algorithms for predicting reputability. Data augmentation methods like SMOTE, ADASYN, and GANs address class imbalance, ensuring equal representation of illicit and reputable contracts. Opcode embeddings train and compare models such as LightGBM, XGBoost, and CatBoost. The hypothesis that data augmentation improves the prediction of illicit contracts is assessed using recall and F1-score, quantifying the effect of balanced datasets on boosting algorithm performance.

Experiment 2: Multimodal Anomaly Detection for Smart Contracts

Addressing Objective 2, this experiment integrates bytecode embeddings with transactional data using multimodal data fusion techniques. The aim is to detect anomalous patterns that signal illicit behaviour. CNN and LSTM-based autoencoders generate reconstruction errors for transactional data, which are combined with static features in a multimodal framework. The hypothesis that fusing static and transactional data enhances detection accuracy is evaluated by comparing the recall and F1-score of multimodal models against those using single data types. These metrics validate the advantage of multimodal fusion in improving sensitivity to anomalous smart contract behaviours.

Experiment 3: Long-Term Temporal Analysis of Smart Contract Reputability

Aligned with Objective 3, this experiment uses LSTM-based sequence models to analyse long-term trends in reputability scores derived from Experiment 2. By modelling gradual changes, it identifies emerging risks or sustained performance patterns. The hypothesis that LSTMs capture long-term trends is evaluated using MAE, MSE, and cross-correlation analysis to assess accuracy and temporal alignment. These metrics validate the model's ability to differentiate meaningful trends from short-term noise in contract reputability.

Together, these experiments form a comprehensive framework for assessing and predicting smart contract reputability using multimodal and temporal approaches. ility using a multimodal and temporal approach.

5 Evaluation

In this chapter, we assess the performance of our proposed models for detecting anomalous behaviour in smart contracts and predicting their reputability over time. The evaluation focuses on comparing the results of various data augmentation techniques alongside model architectures such as Gradient Boosting machines, LSTM-based and CNN-based autoencoders.

5.1 Experiment 1: Boosted Embedding-based Bytecode Analysis

Our first experiment addressing Objective 1 in this research involved evaluating the impact of various data augmentation techniques on the prediction of illicit smart contracts. Due to the inherent imbalance in the dataset, where the number of illicit contracts is significantly lower than reputable ones, we applied three widely used augmentation techniques: SMOTE, ADASYN, and GANs. The primary goal of this experiment was to assess how effectively these techniques improve the model's ability to detect illicit contracts by increasing the representation of the minority class (illicit contracts) and enhancing overall predictive model performance.

We first trained a baseline model on the original, imbalanced dataset using LightGBM. We then augmented the minority class using SMOTE, ADASYN, and GANs, generating synthetic samples to create a more balanced dataset. The models were re-trained on these augmented datasets, and their performances were evaluated using key classification metrics, including recall, precision, and F1-score.

The performance of the models on the original dataset and each augmented dataset was compared to determine the impact of data augmentation on the detection of illicit smart contracts. Special attention was given to the recall metric, as correctly identifying illicit contracts is critical in this context.

5.1.1 Data Augmentation with SMOTE and ADASYN

As discussed in the previous section, the data is heavily imbalanced; to address this imbalance, we applied three widely used augmentation techniques: SMOTE, ADASYN, and GANs. The primary goal was to assess how these techniques improve the model's ability to detect illicit contracts by increasing the representation of the minority class (illicit contracts) and enhancing overall model performance.

Initially, the dataset contained 191 illicit smart contracts and 3085 reputable smart contracts. In order to visualize the opcode embeddings by the category of smart

contract, we use the t-SNE plot shown in Figure 5.1.

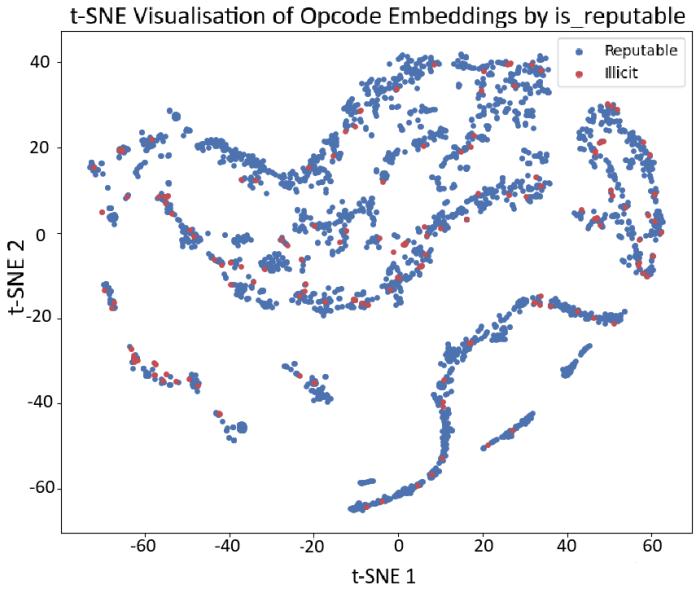


Figure 5.1 t-SNE plot of opcode embeddings coloured by label. The blue points indicate Reputable smart contracts meanwhile the red points represent Illicit smart contracts

The t-SNE visualization of opcode embeddings for the dataset reveals several key observations. As shown in Figure 5.1, the plot indicates distinct clusters of points, suggesting that the opcode embeddings effectively capture meaningful differences between reputable and illicit contracts. There is noticeable separation between the blue (reputable) and red (illicit) points, which implies that the embeddings have potential utility in distinguishing between the two classes. However, some overlap between the clusters is evident, pointing to areas where the embeddings may struggle to differentiate between reputable and illicit contracts due to potential similarities in opcode sequences.

Once the dataset has been split into train and test, we apply SMOTE and ADASYN on the train set. After applying the augmentation techniques, the distributions of illicit and reputable contracts are summarized in Table 5.1. Both SMOTE and ADASYN were used to balance the dataset by increasing the number of illicit contracts to match the number of reputable ones. Specifically, SMOTE and ADASYN both resulted in 2160 illicit contracts, while the original dataset had only 132 illicit contracts.

Technique	Total Contracts	Illicit Contracts	Reputable Contracts
Original	2376	132	2161
SMOTE	4320	2160	2160
ADASYN	4320	2120	2160

Table 5.1 Distribution of Contracts post Resampling Techniques on Training Set

The optimal hyperparameters for the baseline LightGBM model on the different

datasets are detailed in Table 5.2. Notably, the model trained on the original dataset used a lower learning rate and fewer estimators compared to the models trained on the SMOTE and ADASYN datasets. This adjustment was made to account for the distinct characteristics of each dataset.

Dataset	Learning Rate	Max Depth	n_estimators
Original	0.01	2	100
SMOTE	0.2	8	300
ADASYN	0.2	8	300

Table 5.2 Optimal Hyperparameters for LGBM Models on Different Datasets

The performance of the LightGBM model, evaluated on recall, f1-score, and accuracy for illicit contracts, is summarized in Table 5.3. The original dataset's performance shows a recall of 0.000, indicating that the model could not detect illicit contracts due to the severe class imbalance. After augmentation with SMOTE, recall increased to 0.4310, and with ADASYN, it increased to 0.3793. This improvement demonstrates that both SMOTE and ADASYN effectively enhanced the model's ability to identify illicit contracts.

Dataset	Recall	F1-Score	Accuracy
Original	0.000	0.970	0.941
SMOTE	0.431	0.427	0.931
ADASYN	0.379	0.355	0.919

Table 5.3 Performance of LGBM Models on Different Datasets

Although recall improved with resampling techniques, as shown in Table 5.3, the f1-score and accuracy for the datasets augmented using SMOTE and ADASYN decreased slightly compared to the original dataset. It is important to note that while the overall accuracy across all datasets remains high, this metric alone does not fully reflect model performance. The high accuracy achieved by the model is primarily attributed to the overwhelming presence of reputable contracts in the dataset, which effectively masks the model's struggles in accurately identifying illicit contracts.

To illustrate this issue further, we present the ROC-Area Under the Curve (AUC) curves for each dataset. These curves demonstrate that while resampling techniques have enhanced performance to some extent, the model still struggles significantly with detecting illicit smart contracts.

Moreover, while this approach increases the representation of illicit contracts, it has some inherent limitations when applied to complex, high-dimensional data like smart contracts. SMOTE tends to oversimplify by generating synthetic samples that are linearly interpolated between real samples, which may fail to capture the nuanced and

highly varied behaviours present in the illicit contracts. This is evident from the ROC-AUC curves, where SMOTE-augmented models showed improvement in recall but still displayed a significant drop in specificity (high false positive rates).

Similarly, ADASYN introduces variability by focusing on difficult-to-classify samples, which may not fully capture the underlying patterns of illicit contracts. This variability explains the moderate increase in recall but the limited improvement in F1-score, as seen in the ROC-AUC curves.

Despite the improvements, the ROC-AUC curves reveal that the model's capability to distinguish between illicit and reputable contracts remains insufficient, highlighting the need for more effective techniques or further adjustments to enhance detection of the minority class.

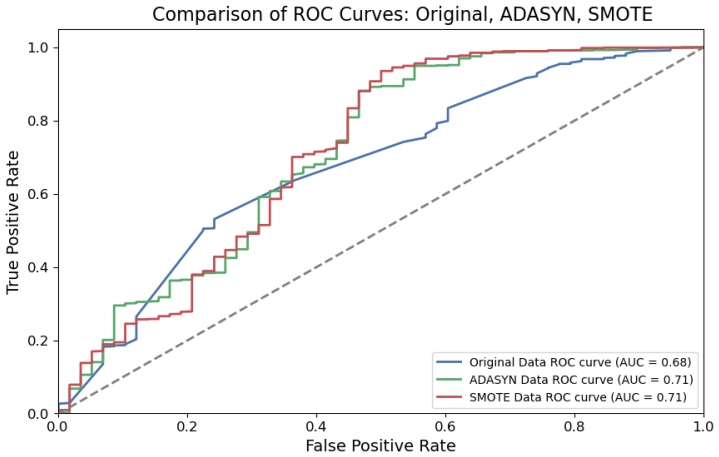


Figure 5.2 ROC Curve Comparison for LGBM Models Trained on Different Datasets.

5.1.2 Data Augmentation with GANs

Despite the improvements in model performance achieved with SMOTE and ADASYN, the recall for detecting illicit smart contracts remained limited, capped at 0.43. To further address this class imbalance, we applied Generative Adversarial Networks (GANs) for data augmentation, focusing only on the minority class, i.e., illicit contracts. This section details the results from using GAN-based data augmentation, including the performance of the GAN and the impact of synthetic data on model evaluation.

The GAN was trained with the hyperparameters listed in Table 5.4 defined in Section 4.3.2. The best hyperparameters are defined in Table 5.4.

Hyperparameter	Value
Input Dimension (Generator)	200
Hidden Units (Both Networks)	256
Dropout Rate	0.2
Batch Size	128
Learning Rate (Generator)	0.0001
Learning Rate (Discriminator)	0.0005

Table 5.4 Optimal Hyperparameters for GAN Training

The training process spanned 10,000 epochs, during which we closely monitored the loss dynamics for both the Generator and Discriminator. The loss curves, illustrated in Figure 5.3, reveal several insights.

Initially, the discriminator's loss decreases as it becomes proficient in distinguishing real from generated data. However, as the generator improves, the discriminator's loss begins to fluctuate before stabilizing. Conversely, the generator's loss starts high but decreases over time, reflecting its progress in generating realistic samples. The eventual equilibrium between the two losses suggests that the GAN has converged, producing high-quality synthetic data.

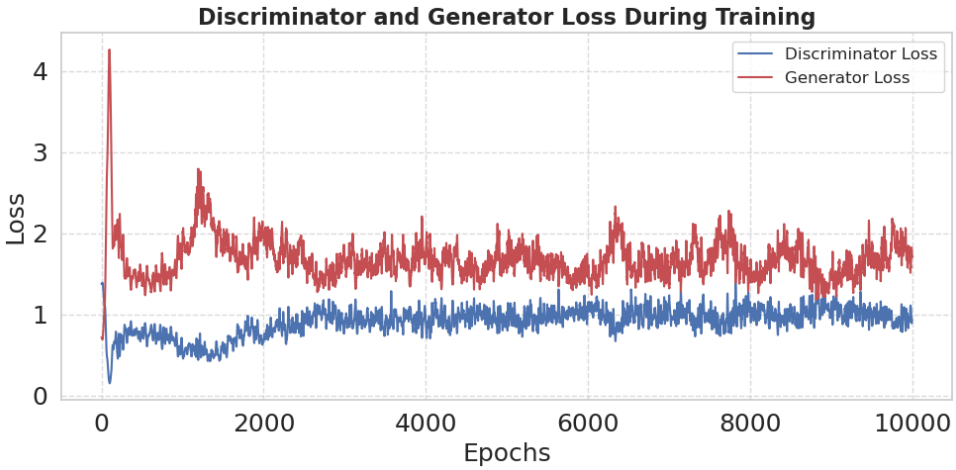


Figure 5.3 Loss Curves for Generator and Discriminator during GAN Training

To evaluate the impact of different batch sizes on data generation, we examined PCA and t-SNE visualizations for batch sizes of 32, 64, and 128. These plots are presented in Figure 5.4:

For batch size 128, the PCA plot shows a clear separation between real and synthetic data points, indicating that the generated data is distinct from the real data. The t-SNE plot reveals that while synthetic data follows the general structure of the real data, it still forms separate clusters, especially in denser regions.

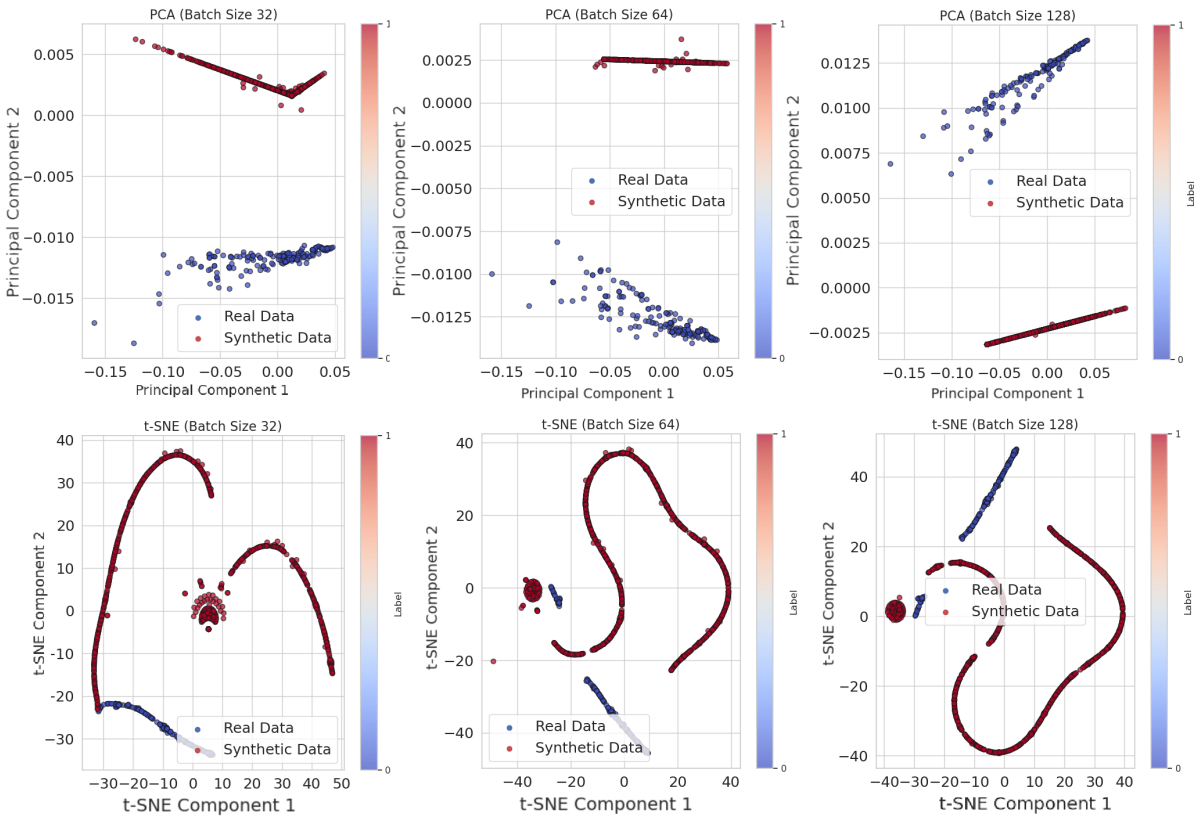


Figure 5.4 PCA and t-SNE plots for batch sizes of 32, 64, and 128. The red dots represent synthetic samples, while the blue dots represent real samples.

With batch size 64, the PCA plot shows a similar separation but with increased overlap compared to batch size 128. The t-SNE visualization indicates that synthetic data is more aligned with real data, particularly in the outer regions, although some clusters remain distinct.

For batch size 32, the PCA plot demonstrates the most overlap between real and synthetic data, with less distinct separation compared to larger batch sizes. The t-SNE plot shows that synthetic data closely follows the real data's path, with fewer outliers and clusters, suggesting better generalization and capturing of the underlying data distribution with smaller batch sizes.

Overall, these results indicate that smaller batch sizes, such as 32, lead to greater overlap between real and synthetic data, which is desirable for balancing datasets and reducing classification bias. However, batch sizes like 128 offer clearer separation, which is ideal in our case of distinguishing illicit cases.

To further assess the quality of the synthetic data generated by our GANs, we performed a series of statistical analyses and comparisons with the real data.

Figure 5.5 presents the Kernel Density Estimation (KDE) plots for real and synthetic opcode embeddings, offering insight into the similarity between the two distributions. The significant overlap between the real and synthetic data suggests that the

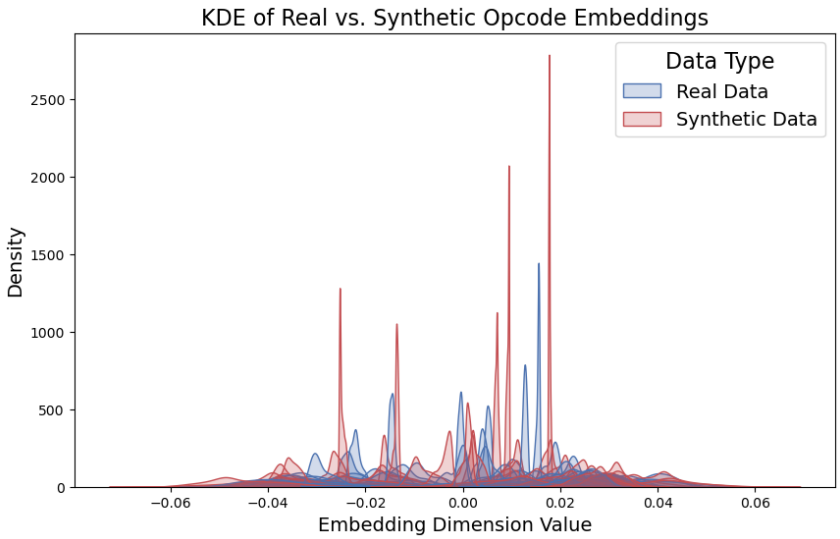


Figure 5.5 KDE of Real vs. Synthetic Opcode Embeddings

generative model was successful in capturing the underlying distribution of opcode embeddings. Both the real and synthetic distributions exhibit sharp density peaks around zero, indicating that many embeddings are concentrated near this value. This similarity in peak density is a promising indicator that the synthetic data effectively replicates the characteristics of the real dataset in terms of core values.

However, upon closer inspection, the synthetic data shows slightly heavier tails compared to the real data, suggesting that it may introduce more extreme values. This divergence could indicate that while the generative model captures the central tendencies well, it potentially exaggerates the occurrence of outlier values. Moreover, although there is notable overlap between the two distributions, there are subtle differences in mode locations, possibly reflecting minor variations in the patterns captured by the synthetic embeddings.

Next, we assessed whether there is a significant difference between the distributions of real and synthetic data. To do this, we first evaluated if both samples follow a normal distribution by presenting the histograms and Q-Q plots in Appendix C and performing a Shapiro-Wilk test. The results showed a p-value of 0.00 for both the real and synthetic samples, leading us to reject the null hypothesis and conclude that neither the real nor the synthetic samples are normally distributed.

Thus, we performed the Mann-Whitney U test to compare the distributions of real and synthetic data. The hypotheses for this test are:

Null Hypothesis (H_0): There is no significant difference between the distributions of the real and synthetic data.

Alternative Hypothesis (H_1): There is a significant difference between the distributions of the real and synthetic data.

Upon performing the test, the p-value was **0.299** which is greater than 0.05, our chosen level of significance, thus, we fail to reject the null hypothesis, suggesting that there is no statistically significant difference between the distributions.

Algorithm	Accuracy
Mean Absolute Deviation	0.0022
Variance Ratio	0.4655
Correlation Coefficient	0.9463
FID	0.0005

Table 5.5 Comparison Metrics Between Real and Synthetic Data

To further assess the similarity between the real and synthetic data, we utilized several key metrics, as summarized in Table 5.5. These metrics provide insight into various aspects of the data distributions. The MAD indicates a very small average discrepancy, suggesting that the synthetic data closely mirrors the real data. Moreover, the Variance Ratio of **0.465** indicates that the variance of the synthetic data is approximately half that of the real data, reflecting a narrower spread in the synthetic dataset.

Additionally, the high Correlation Coefficient of 0.946 demonstrates a strong degree of similarity between the distributions of the real and synthetic data. The low FID score indicates that the synthetic data closely matches the real data distribution in terms of feature statistics.

These metrics collectively support the conclusion that the synthetic data generated by the GAN is highly comparable to the real data in terms of distribution characteristics even though there are subtle differences in distribution and normality.

Dataset	Recall	F1-Score	Accuracy
Original	0.000	0.970	0.941
SMOTE	0.431	0.427	0.931
ADASYN	0.379	0.355	0.919
GAN	0.942	0.933	0.977

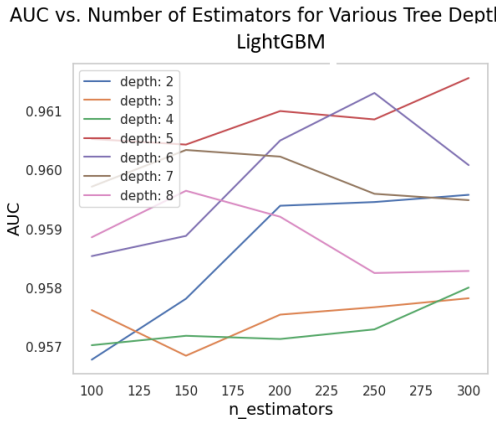
Table 5.6 Performance of LGBM Models on Different Datasets

The results presented in Table 5.6 show the performance of LightGBM models on datasets generated by different techniques: Original, SMOTE, ADASYN, and GANs. The GANs-generated data achieves the highest recall and F1-score, demonstrating its ability to closely match the real data distribution. While SMOTE and ADASYN show reasonable performance, their recall and F1-scores are significantly lower. This is likely because SMOTE and ADASYN generate synthetic samples by interpolating between existing instances in the feature space, which may not capture the complex, non-linear structure of high-dimensional embeddings derived from smart contract bytecode.

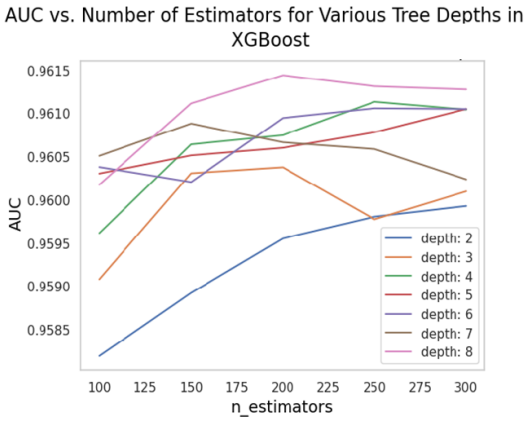
As a result, the synthetic vectors may not represent valid or meaningful behaviour in this space. In contrast, GANs learn the underlying distribution of the minority class and generate entirely new examples that better reflect its variability and semantic richness. This enables the creation of a more discriminative training set, allowing classifiers trained on GAN-augmented data to generalize more effectively, as confirmed by the improved evaluation metrics.

5.1.3 Evaluation of Boosting Algorithms

To comprehensively evaluate the boosting algorithms, we prioritized optimizing the ROC AUC score, which provides a robust measure of model performance. The optimal hyperparameters for each boosting algorithm are detailed in Appendix A.



(a) LightGBM Hyperparameter Tuning



(b) XGBoost Hyperparameter Tuning

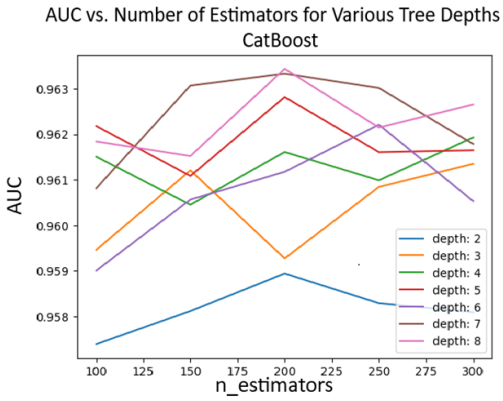
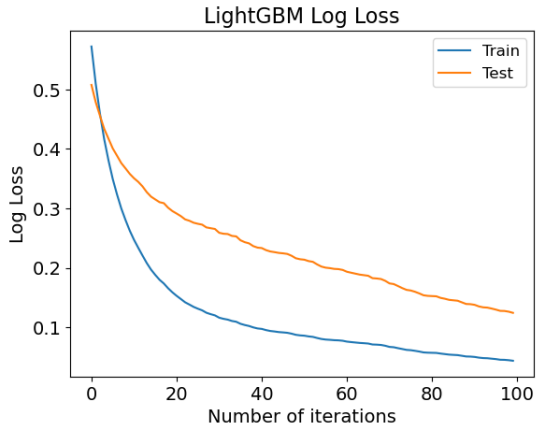


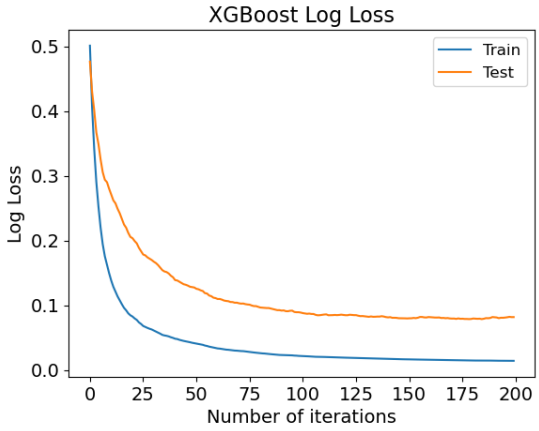
Figure 5.6 ROC AUC versus Number of Estimators for Various Tree Depths in LightGBM (left) and XGBoost (right) and CatBoost (bottom) across 5-Fold Cross Validation.

Figure 5.6 presents the ROC-AUC scores as a function of the number of estimators for both LightGBM and XGBoost, with varying tree depths. These plots illustrate the impact of hyperparameter tuning on model performance. LightGBM achieves an optimal ROC AUC score of **0.961** with a maximum depth of 5 and 300 estimators, while

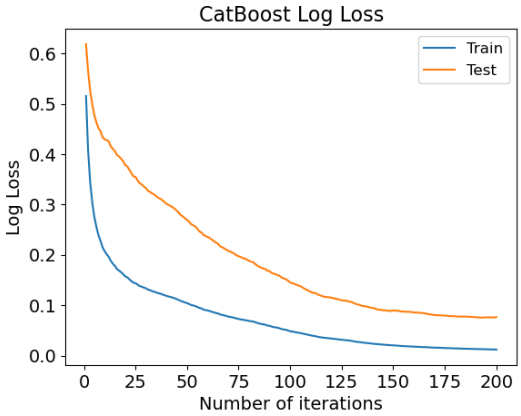
XGBoost demonstrates a similar ROC AUC of **0.961** with a maximum depth of 8 and 200 estimators. We further evaluated model performance by analysing log loss, as shown in Figure 5.7.



(a) LightGBM Log Loss



(b) XGBoost Log Loss



(c) CatBoost Log Loss

Figure 5.7 Average logarithmic loss, evaluated through 5-Fold Cross Validation, in relation to the number of iterations for LightGBM (left) and XGBoost (right).

Figure 5.7 shows the log loss curves for LightGBM and XGBoost. For LightGBM, the log loss stabilizes after approximately 200 iterations, indicating that the model's performance has reached a plateau and further training may offer limited benefits. In contrast, XGBoost exhibits a rapid decrease in training loss, but the validation loss starts to increase after about 150 iterations, suggesting potential overfitting.

To address overfitting and ensure robust performance, we employed K-Fold Cross Validation with 5 folds. This approach systematically divides the dataset into five subsets, using each as a validation set while the others serve as the training set. The performance metrics for each boosting algorithm are summarized in Table 5.7.

Table 5.7 summarizes the performance metrics for each algorithm. LightGBM achieves the highest accuracy of **97.67%** and a recall of **0.942** for the illicit class, demonstrating its superior capability in both synthetic and real data contexts. Although XG-

Algorithm	Recall	F1-Score	Accuracy
LightGBM	0.942	0.933	97.67%
CatBoost	0.937	0.930	97.58%
XGBoost	0.942	0.928	97.49%

Table 5.7 Performance Metrics for Boosting Algorithms

Boost and CatBoost also perform well, LightGBM's higher accuracy and F1-score underscore its overall efficacy.

Table 5.8 presents a summary of the best hyperparameters found for each model and detailed classification reports and classification errors for each model are presented as supplementary results in Appendix C.

Hyperparameter	LightGBM	XGBoost	CatBoost
Learning Rate	0.1	0.2	0.1
Maximum Depth	5	8	8
Subsample	0.5	0.5	0.5
Regularization Alpha	0.1	0.1	0.1
Regularization Lambda	0.01	0.01	0.01
Number of Estimators	300	200	200

Table 5.8 Hyperparameters for Boosting Algorithms

In summary, the evaluation of our boosting algorithms underscores the effectiveness of data augmentation in enhancing the prediction of illicit smart contracts.Initially, the imbalanced dataset resulted in a recall of 0.00 for illicit smart contracts. SMOTE and ADASYN improved recall to 0.431 and 0.379, respectively. The greatest improvement was achieved with GAN-augmented data, reaching a recall of **0.942**. This substantial enhancement validates our hypothesis that data augmentation can significantly improve the prediction of illicit smart contracts. The superior performance of models trained on GAN-augmented data, in particular, confirms that advanced augmentation techniques are crucial for effectively identifying illicit smart contracts in a dataset that initially suffers from severe class imbalance.

Benchmarking on Ponzi Dataset

Following the successful evaluation of LightGBM as the best-performing model in Experiment 1, we benchmarked it on the Ponzi dataset from [97] which includes contracts labelled as either Ponzi schemes or normal (non-Ponzi). This benchmarking was conducted to assess the generalizability of our model, trained on GAN-augmented opcode embeddings, against a real-world dataset of Ponzi schemes on Ethereum. The bench-

mark dataset comprises 3,769 contracts, of which 200 are labelled Ponzi schemes and 3,569 as normal contracts.

In comparison to the original model in [97] which achieved an accuracy of 0.9956 with a recall of 0.960 for Ponzi schemes and an F1-score of 0.860, our LightGBM model demonstrated an accuracy of 0.994 on the benchmark dataset. Although our model's accuracy is slightly lower than the 0.996 reported in [97], it shows notable improvements in precision, recall, and F1-score for the Ponzi class. Specifically, our model achieved a precision of 0.949, recall of 0.930, and an F1-score of 0.939. These results are detailed in Table 5.9. The precision of 0.9490 indicates a substantial reduction in false positives compared to the benchmark results from [97], which had a precision of 0.940 for Ponzi schemes. Furthermore, while the recall of 0.930 is somewhat lower than 0.960 reported in [97], it still reflects strong performance, demonstrating that our model can effectively detect Ponzi schemes while maintaining high precision

Model	Class	Precision	Recall	F1-Score	Support
LightGBM	**Ponzi**	**0.949**	**0.930**	**0.939**	200
	Normal (Non-Ponzi)	0.996	0.997	0.997	3569
Accuracy		0.994			3769
Macro Avg	0.974	0.964	0.968	3769	
Weighted Avg	0.994	0.994	0.994	3769	
XGBoost [97]	**Ponzi**	**0.940**	**0.810**	**0.860**	200
	Normal (Non-Ponzi)	0.990	0.990	0.990	3569
Accuracy		0.990			3769
Macro Avg	0.920	0.810	0.860	3769	
Weighted Avg	-	-	-	3769	

Table 5.9 Performance Metrics of LightGBM on the Benchmark Dataset compared with [97]

These results suggest that our LightGBM model, leveraging GAN-augmented opcode embeddings, provides a competitive alternative to the approach used in [97] by achieving comparable recall (0.930) and accuracy (0.994), while improving the F1-score (0.939) for Ponzi scheme detection. The increase in recall from 0.810 in the benchmark model to 0.930 highlights the model's improved ability to identify Ponzi schemes, which is crucial in minimizing missed detections in real-world applications. Additionally, the enhanced F1-score indicates a balanced performance, effectively weighing both precision and recall. This improvement over the original model underscores the value of GAN-based augmentation, enabling the model to capture broader patterns of illicit behaviour that generalize well to external datasets.

5.1.4 Discussion and Further Insights

The first experiment aimed to acquire, preprocess, and analyse a balanced dataset of smart contracts to evaluate the performance of boosting algorithms in predicting reputability based on bytecode embeddings therby addressing objective 1 (Section 1.3). This was a foundational step toward our overarching goal of developing a robust framework for reputability prediction and anomaly detection in smart contracts. The results achieved through this initial stage offer significant insights into the capabilities and limitations of machine learning models in this context.

Our findings confirmed that boosting algorithms, specifically LightGBM, outperformed other algorithms such as XGBoost and CatBoost in terms of both accuracy and recall for predicting the reputability of smart contracts. The optimal model, trained on GAN-augmented data, demonstrated a strong performance, with LightGBM achieving an impressive accuracy of 97.67% and a recall of 0.942 for the illicit class. This result underscores the efficacy of data augmentation techniques, particularly GANs, in addressing class imbalance, a critical challenge in the field of blockchain analysis. The improved recall rate shows the model's ability to identify potential illicit contracts, aiding in proactive risk mitigation.

Benchmarking also revealed our model's strength. The combination of GAN-augmented opcode embeddings and the LightGBM model proved effective in identifying illicit contracts, showcasing competitive performance metrics when compared to the original XGBoost model in [97]. This comparison demonstrates that our approach not only matches but can potentially enhance existing methodologies in the field. The ability of our model to generalize to unseen data while maintaining high recall and F1-score highlights its potential as a robust tool for reputability assessment in blockchain environments. Such benchmarking confirms that our method is well-positioned for further integration with other data modalities and analytical techniques in subsequent phases of the study.

In the broader context of our study's aim, these results validate that embedding-based bytecode analysis, when combined with effective data augmentation strategies, can provide a reliable basis for predicting the reputability of smart contracts. The findings lay the groundwork for the integration of bytecode embeddings with transactional data in subsequent experiments. This is essential for building a comprehensive framework capable of assessing reputability and detecting anomalies over time, which aligns with the project's objective of multimodal data fusion.

The performance metrics observed not only highlight the predictive potential of boosting algorithms in this domain but also suggest pathways for further analysis, such as understanding feature importance in model decisions and exploring interpretability to enhance trust in the framework. Moreover, these initial results emphasize the necessity

of addressing data imbalance through advanced augmentation methods, as traditional approaches such as SMOTE and ADASYN, while useful, did not yield as significant an improvement as GAN-based techniques.

Ultimately, this experiment has laid a strong foundation for the subsequent objectives of the thesis. By evaluating the performance of boosting algorithms in predicting smart contract reputability based on embedding-based bytecode analysis, we have gained crucial insights into how these features can contribute to more comprehensive reputability assessments. This initial success not only highlights the effectiveness of GAN-augmented opcode embeddings in boosting model performance but also supports the transition to integrating static and transactional data in the next phase of the research, which is pivotal for advancing toward the overarching goal of developing a multimodal framework for dynamic, real-time evaluation of smart contract reputability and anomaly detection. Through this experiment, we have successfully achieved Objective 1 by demonstrating that boosting algorithms, particularly when trained with GAN-augmented opcode embeddings, can effectively predict the reputability of smart contracts based on embedding-based bytecode analysis. The strong performance metrics confirm our approach as a reliable tool for reputability assessment, and integrating these findings with transactional data will enhance the predictive framework's robustness and scope.

5.2 Experiment 2: Multimodal Anomaly Detection for Smart Contracts

In this experiment, we address Objective 2 (Section 1.3) of the thesis by evaluating multimodal anomaly detection techniques applied to smart contracts. The main objective is to assess the effectiveness of combining temporal and spatial features from transaction sequences to improve anomaly detection accuracy. To this end, we employ CNN-based autoencoders and compare their performance against an LSTM-based autoencoder.

The experimental setup includes training the CNN and LSTM autoencoders on transaction sequences. These models are evaluated based on their ability to capture and reconstruct transaction patterns, with the reconstruction error serving as the key metric for anomaly detection. The LSTM autoencoder focuses on long-term temporal dependencies, while the CNN autoencoder targets local patterns within the data. By evaluating and comparing the performance of these two models, we can gain insights into which model is better at anomaly detection.

5.2.1 Transaction-Level Anomaly Detection

We first evaluate the model by excluding the opcode embeddings from the transactions, effectively analysing the transactional data in isolation without integrating bytecode embeddings. The dataset consists of 820,920 transactions from reputable smart contracts (after outlier removal) and 6,761 transactions from illicit smart contracts.

The Convolutional Autoencoder (CAE) was evaluated for anomaly detection using several metrics to assess its performance. The analysis included the distribution of reconstruction errors, validation loss trends, threshold impact on True Positives Rate (TPR) and False Positives Rate (FPR), and the classification metrics.

Figure 5.8 illustrates the density distribution of reconstruction errors between reputable and illicit smart contracts.

The median reconstruction error for reputable contracts is significantly lower, measured at 0.04, indicating that the CAE was able to efficiently reconstruct these contracts. The distribution shows a steep decline after the peak, reflecting that most reputable contracts tend to have low reconstruction errors. This suggests that the model is highly effective at learning the underlying structure of reputable contracts.

In contrast, the illicit contracts exhibit a higher median reconstruction error of 0.073. The distribution for illicit contracts shows a less steep decline and is centered around higher reconstruction error values, suggesting that the CAE struggles more to accurately reconstruct illicit contracts. This increase in reconstruction error for illicit contracts aligns with our hypothesis that illicit contracts are more anomalous in nature, and thus more difficult for the model to learn and reconstruct accurately.

The distinct separation between the two distributions highlights the effectiveness of using reconstruction error as a feature for distinguishing between reputable and illicit contracts. The higher median for illicit contracts supports the claim that reconstruction error can serve as an informative signal for identifying non-reputable behaviour in smart contracts. The visualization further strengthens the case for applying threshold-based anomaly detection techniques, where illicit contracts can be flagged based on their significantly higher reconstruction error.

Furthermore, to evaluate the impact of different batch sizes on data generation, we examine the loss curves across different batch sizes presented in Figure 5.9. The training and validation loss curves for batch sizes 16, 32, and 64 provide important insights into the model's generalization and stability. For batch size 16, the training and validation losses decrease steadily and consistently throughout the epochs, with minimal fluctuations. The small gap between the two losses indicates strong generalization, with little to no overfitting. This stable behaviour suggests that the model can learn effectively with batch size 16, making it a promising candidate.

In contrast, batch size 32 exhibits noticeable fluctuations in the validation loss,

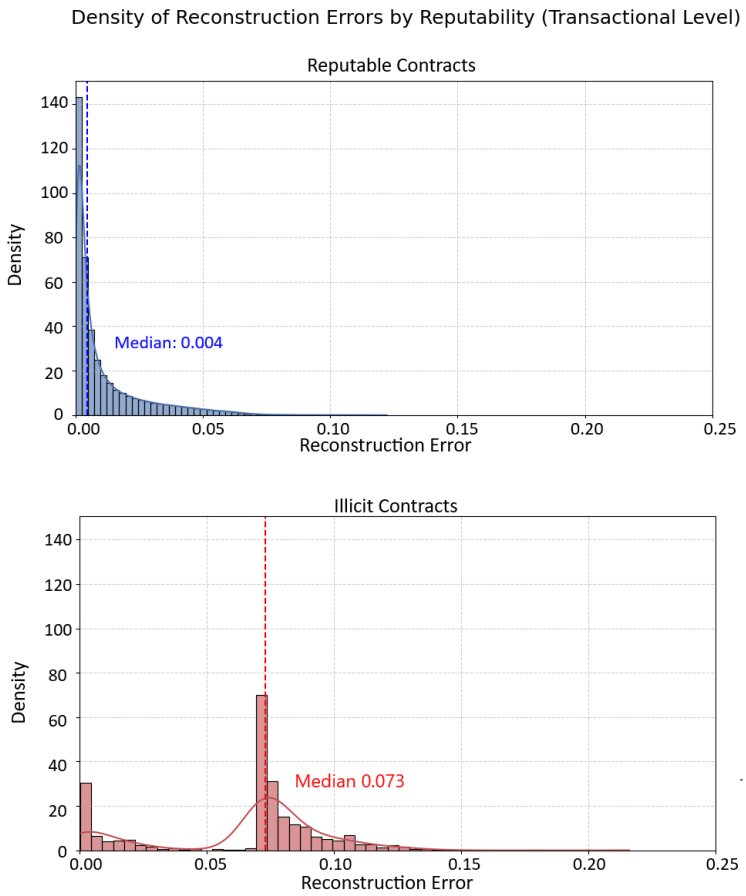
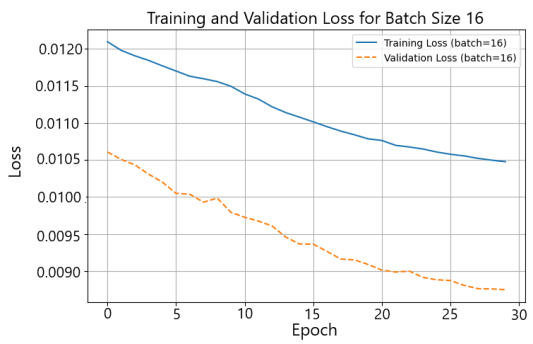


Figure 5.8 Distribution of Reconstruction Errors for the Convolutional Autoencoder (CAE). The graph shows the frequency of different reconstruction error values.

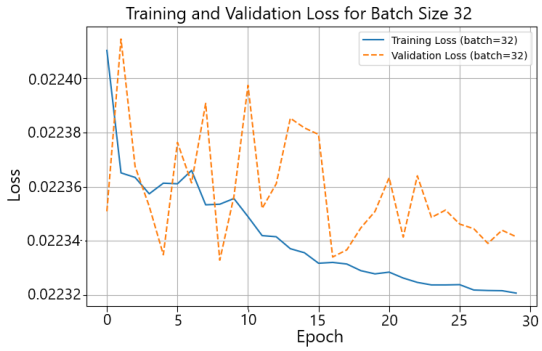
despite a steady decrease in training loss. The oscillations in validation performance suggest some instability during training, potentially caused by overfitting or noisy updates. While the model still converges, this instability makes batch size 32 less reliable compared to the smoother performance seen with batch size 16.

Finally, batch size 64 demonstrates a relatively smooth decrease in both training and validation loss, similar to batch size 16. However, the initial validation loss is higher, and it takes more epochs to reach a lower value. Although the performance is acceptable, batch size 64 does not achieve the same low validation loss as batch size 16 and may require more training to fully converge.

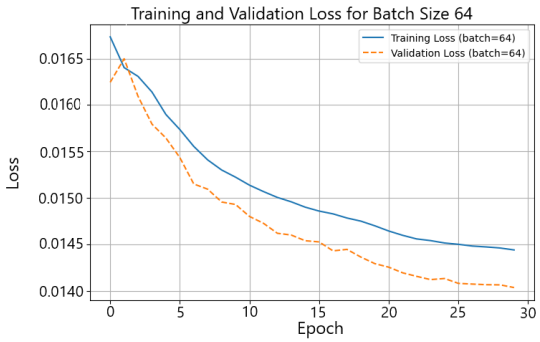
To further illustrate the tuning process, Table 5.10 summarizes the key hyperparameters and the average validation loss across different batch sizes. This highlights the relationship between the configurations and their corresponding performance. The results of the hyperparameter tuning reveal that batch size 16 achieved the best performance with an average validation loss of 0.01059. These results suggest that a smaller batch size is more conducive to stable and effective learning in this context, likely due to more frequent updates and smoother convergence during training.



(a) Training and Validation for Batch Size 16



(b) Training and Validation for Batch Size 32



(c) Training and Validation for Batch Size 64

Figure 5.9 Training and Validation Loss Curves for the Convolutional Autoencoder (CAE) across batch sizes of 16, 32, and 64. The graph shows the stability of loss values over epochs.

Batch Size	Filters 1	Filters 2	Learning Rate	Activation	Avg Validation Loss
16	64	8	0.00297	Tanh	0.01059
32	64	24	0.00195	Tanh	0.02172
64	16	32	0.00076	ReLU	0.01606

Table 5.10 Summary of key hyperparameters and average validation loss for different batch sizes across five folds.

In comparison, batch size 32 showed a higher average validation loss of 0.02172, despite employing a similar number of filters in the first layer (64) and a slightly more complex architecture with 24 filters in the second layer. The tanh activation function and a learning rate of 0.00195 were used for this batch size, but the increased batch size appeared to introduce instability in the model's ability to generalize, as evidenced by higher validation loss.

Finally, batch size 64 also performed less favorably than batch size 16, with an average validation loss of 0.01606. While the learning rate was the smallest among the configurations (0.00076), the use of the ReLu activation function combined with 16 filters in the first layer and 32 filters in the second layer did not compensate for the

larger batch size. This suggests that while the model can still generalize reasonably well with a larger batch size, it requires more epochs to converge, and the performance is not as robust as with smaller batch sizes. The full list of hyperparameters can be found in Appendix D.

To further assess the performance of the Convolutional Autoencoder (CAE) in classifying reputable and illicit smart contracts, Table 5.11 summarizes key metrics across various threshold values. These metrics include the Area Under the Curve (AUC), accuracy, and F1-scores for both reputable and illicit contracts, as well as the recall for detecting illicit contracts.

Threshold	AUC	Accuracy	F1-Score (Reputable)	F1-Score (Illicit)	Recall (Illicit)
75	0.9030	0.9155	0.9455	0.8121	0.8816
80	0.9087	0.9401	0.9622	0.8553	0.8553
85	0.9190	0.9564	0.9728	0.8904	0.8553
90	**0.9454**	**0.9673**	**0.9795**	**0.9200**	**0.9079**

Table 5.11 Performance Metrics at Different Thresholds for the CNN-Based Autoencoder. The row in bold indicates the best threshold that results in the best f1-score for the illicit smart contracts.

As demonstrated in Table 5.11, the performance improves steadily as the threshold increases from 75 to 85, with the model achieving its best balance at a threshold of 90. At this point, the model achieves an AUC of 0.9454, an accuracy of 96.73%, and an F1-score of 0.9673 for reputable contracts, alongside a strong F1-score of 0.9200 and recall of 90.79% for illicit contracts.

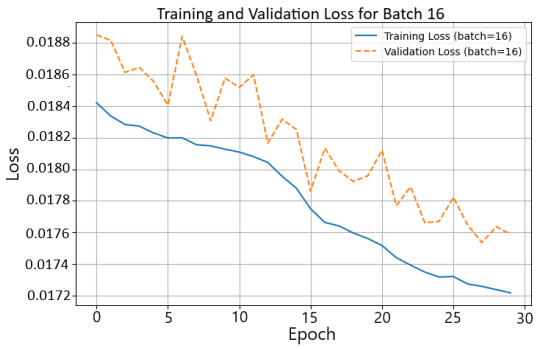
LSTM-Based Autoencoder

We now turn our attention to the LSTM-Based Autoencoder. To ensure fairness, we trained the LSTM-AE on the same dataset as used earlier for the CAE and employed similar hyperparameter tuning as conducted for the CAE. The plots in Figure 5.10 demonstrate the training loss and validation across various batch sizes.

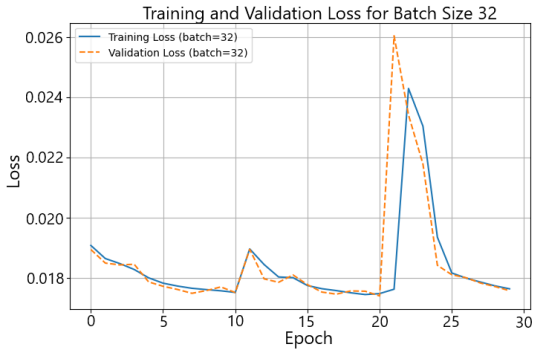
The hyperparameters selected for the LSTM-based autoencoder are summarized in Table 5.12. The complete list of hyperparameters can be found in Appendix D.

Batch Size	Filters 1	Filters 2	Learning Rate	Activation	Avg Validation Loss
16	64	16	0.0003095	ReLU	0.01059
32	64	24	0.00195	Tanh	0.02172
64	16	32	0.00076	ReLU	0.01606

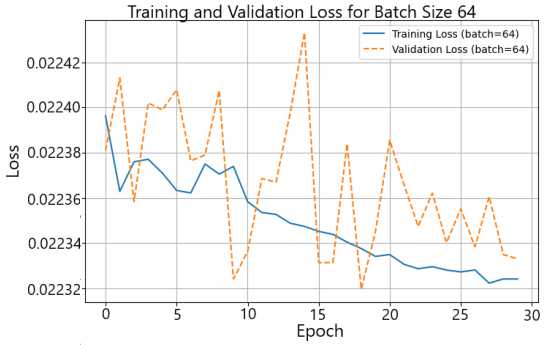
Table 5.12 Summary of Hyperparameters for the LSTM-Based Autoencoder



(a) Training and Validation for Batch Size 16



(b) Training and Validation for Batch Size 32



(c) Training and Validation for Batch Size 64

Figure 5.10 Training and Validation Loss Curves for the LSTM-Based Autoencoder across batch sizes of 16, 32, and 64. The graph shows the stability of loss values over epochs.

The performance of the LSTM-based autoencoder was evaluated using various thresholds, and the results are summarized in Table 5.13. Notably, the best results were achieved at a threshold of 90, demonstrating significant improvements in both recall and overall accuracy.

Threshold	AUC	Accuracy	F1-Score (Reputable)	F1-Score (Illicit)	Recall (Illicit)
75	0.9013	0.9128	0.9675	0.7444	0.8816
80	0.9019	0.9292	0.9617	0.8125	0.8553
85	0.9173	0.9537	0.9628	0.9155	0.8553
90	**0.9387**	**0.9591**	**0.9792**	**0.8861**	**0.9211**

Table 5.13 Performance Metrics at Different Thresholds for the LSTM-Based Autoencoder. The row in bold represents the best threshold that results in the best f1-score for the illicit smart contracts.

When comparing the performance of the LSTM-based autoencoder with that of the CAE, it becomes clear that the CAE outperforms the LSTM model in several key metrics. The CAE achieved a lower median reconstruction error for both reputable and

illicit contracts, indicating a more effective learning of the underlying patterns inherent to these contracts. The CAE's ability to capture local features in transaction data allows for more precise anomaly detection, whereas the LSTM model, while capable of capturing temporal dependencies, struggled to generalize as effectively as the CAE.

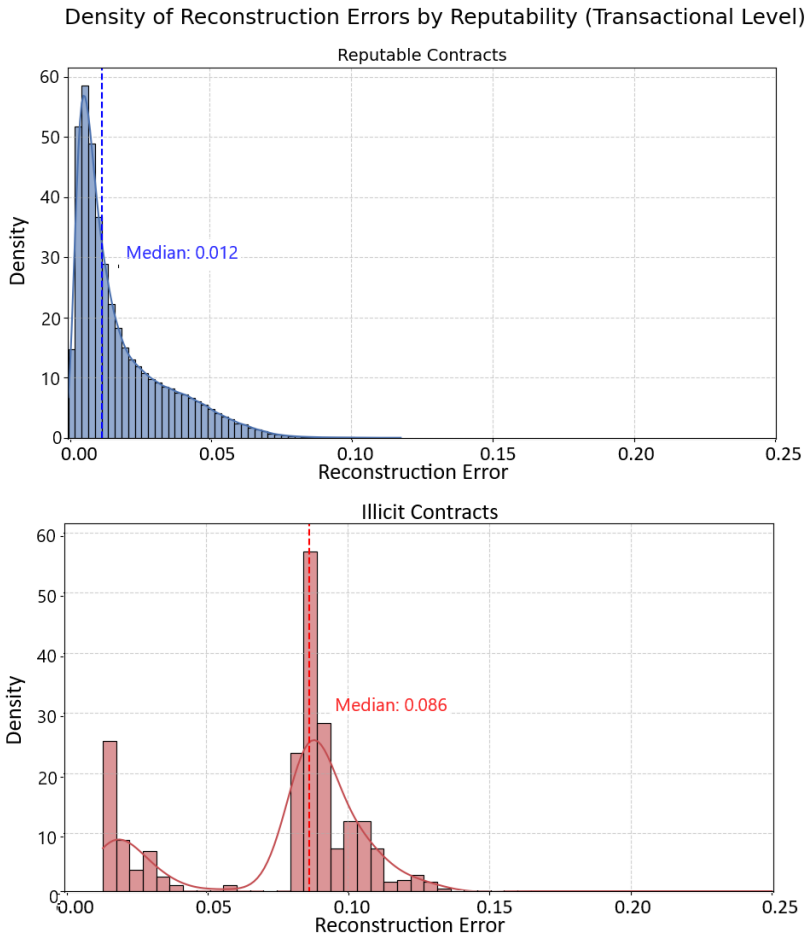


Figure 5.11 Distribution of Reconstruction Errors for the LSTM-Autoencoder. The graph shows the frequency of different reconstruction error values.

The density plots of reconstruction errors in Figure 5.11 from the LSTM-based autoencoder show a clear distinction between reputable (median: 0.012) and illicit contracts (median: 0.086). In comparison, the CNN-based autoencoder exhibits lower reconstruction errors for both classes, indicating better performance in distinguishing reputable from illicit contracts. The CAE's ability to capture relevant features more effectively results in tighter error distributions, demonstrating its superiority over the LSTM model for anomaly detection in smart contracts.

Table 5.14 summarizes the key performance metrics for both models, underscoring the advantages of the CAE approach.

Model	AUC	Accuracy	F1-Score (Illicit)	Recall (Illicit)	Median Reconstruction Error
CAE	0.9454	96.73%	0.9200	90.79%	0.073
LSTM	0.9387	95.91%	0.8861	92.11%	0.086

Table 5.14 Comparison of Performance Metrics between the CAE and LSTM-Based Autoencoder at a Threshold of 90

In conclusion, while both models can detect anomalies, the CAE outperforms the LSTM due to its more efficient reconstruction of transaction patterns. Although LSTMs are valuable for sequential data, they showed limitations in generalizing across the dataset. This highlights the CAE's suitability for detecting complex anomalies associated with transactions in smart contracts.

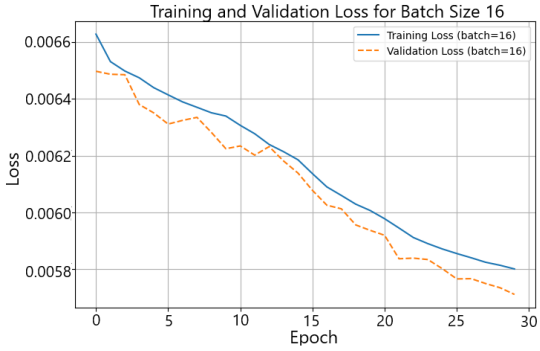
5.2.2 Multimodal Anomaly Detection Model

Building on the insights gained from evaluating individual models, we now explore a more advanced approach to anomaly detection by integrating multiple data modalities. Although the CAE model demonstrated strong performance using only transaction data, further improvements may be possible by incorporating additional data sources. The previous sections showed that the transaction-based CAE was highly effective in detecting anomalies, but complex patterns in smart contract behaviour may not be fully captured through transactions alone.

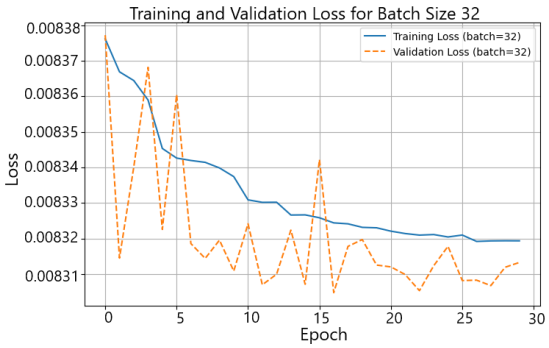
In this section, we extend the anomaly detection approach by integrating opcode embeddings alongside transaction data, creating a multimodal model that leverages both bytecode embeddings and transactional activity. The hypothesis is that this fusion will provide a more comprehensive representation of each contract's behaviour, potentially improving the model's ability to distinguish between reputable and illicit contracts. The following evaluation compares the performance of this multimodal approach against the single-modality CAE to assess whether incorporating multiple data modalities leads to further gains in accuracy, recall, and overall detection capability. Since the CAE model outperformed the LSTM-AE at the transaction-level, we focus primarily on the CAE.

The training and validation loss curves for the multimodal CAE reveal similar trends to the transaction-only model. As shown in Figure 5.12, batch size 16 consistently provides the most stable learning, with minimal fluctuations in validation loss across epochs. The model generalizes well, and the smooth convergence indicates that batch size 16 allows for frequent updates, leading to effective learning of the complex

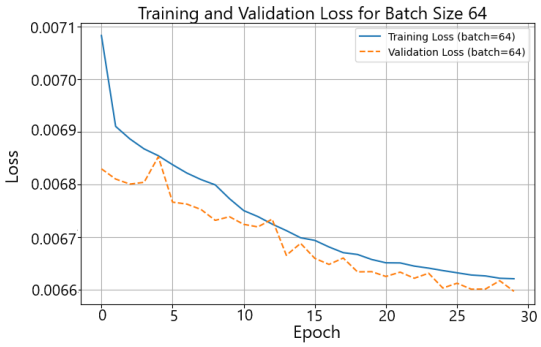
multimodal relationships. This confirms that smaller batch sizes are better suited for handling both transactional and opcode data, achieving lower validation losses.



(a) Training and Validation for Batch Size 16



(b) Training and Validation for Batch Size 32



(c) Training and Validation for Batch Size 64

Figure 5.12 Training and Validation Loss Curves for the Multimodal Convolutional Autoencoder (CAE) across batch sizes of 16, 32, and 64. The graph shows the stability of loss values over epochs.

On the other hand, batch size 32 introduces noticeable fluctuations in validation loss, signalling instability and slower convergence despite using a more complex architecture and the ReLU activation function. Batch size 64, while more stable than 32, still requires more epochs to converge and does not outperform batch size 16. Overall, the results show that batch size 16 remains the optimal choice for multimodal CAE, balancing efficient learning and generalization. This trend aligns with the findings from the transaction-only CAE (Section 5.2.1, further supporting the choice of a smaller batch size for best performance in multimodal tasks.

To further illustrate the tuning process, Table 5.15 summarizes the key hyperparameters and the average validation loss across different batch sizes when using multimodal data (transaction and opcode embeddings). Batch size 16 achieved the lowest average validation loss, with a value of 0.00627. This configuration used 48 filters in the first layer, 24 filters in the second, a learning rate of 0.00121, and ReLU activation. These hyperparameters allowed the model to learn in a stable and efficient manner, supporting effective generalization. The complete list of hyperparameters can be found in

Appendix D.

Batch Size	Filters 1	Filters 2	Learning Rate	Activation	Avg Validation Loss
16	48	24	0.00121	ReLU	0.00627
32	48	16	0.00163	Tanh	0.00810
64	16	8	0.00235	ReLU	0.00675

Table 5.15 Summary of key hyperparameters and average validation loss for different batch sizes across five folds from the Multimodal CAE

Batch size 32, while slightly more complex, with 48 filters in the first layer and 16 in the second, had a higher average validation loss of 0.00810. The use of the Tanh activation function and a learning rate of 0.00163 provided some promise, but the larger batch size appeared to introduce instability, as reflected in the higher validation loss. Batch size 64, which simplified the architecture to 16 filters in the first layer and 8 in the second, also showed a higher validation loss of 0.00675. This suggests that while batch size 64 generalizes reasonably well, batch size 16 remains optimal for the multimodal data fusion architecture.

Next, we analysed the density distribution of reconstruction errors for reputable and illicit contracts, as illustrated in Figure 5.13. For reputable contracts, the median reconstruction error is lower, measured at 0.004, which is significantly smaller compared to the results from the transaction-only CAE model. This lower reconstruction error suggests that the inclusion of opcode embeddings has allowed the model to better capture the underlying structure of reputable contracts, resulting in a more accurate reconstruction and a denser concentration of low error values. The distribution exhibits a steep decline after its peak, indicating that most reputable contracts have very low reconstruction errors.

In contrast, illicit contracts display a higher median reconstruction error of 0.029. Although this value is smaller than the 0.073 observed in the transaction-based model, the distribution remains more skewed towards higher reconstruction errors compared to reputable contracts. This suggests that while the inclusion of opcode embeddings has improved the model's ability to reconstruct illicit contracts, these contracts still present more challenges to the model, resulting in higher reconstruction errors overall.

Given that the average reconstruction error is lower with the multimodal CAE compared to the transaction-only CAE, we conducted statistical tests to assess whether the difference in reconstruction errors between reputable and illicit contracts is significant. First, the Shapiro-Wilk test was applied to check for normality. Since the data was not normally distributed, we proceeded with a one-tailed Mann-Whitney U test to compare the reconstruction error distributions of reputable and illicit contracts.

Null Hypothesis (H_0): There is no significant difference between the distributions of

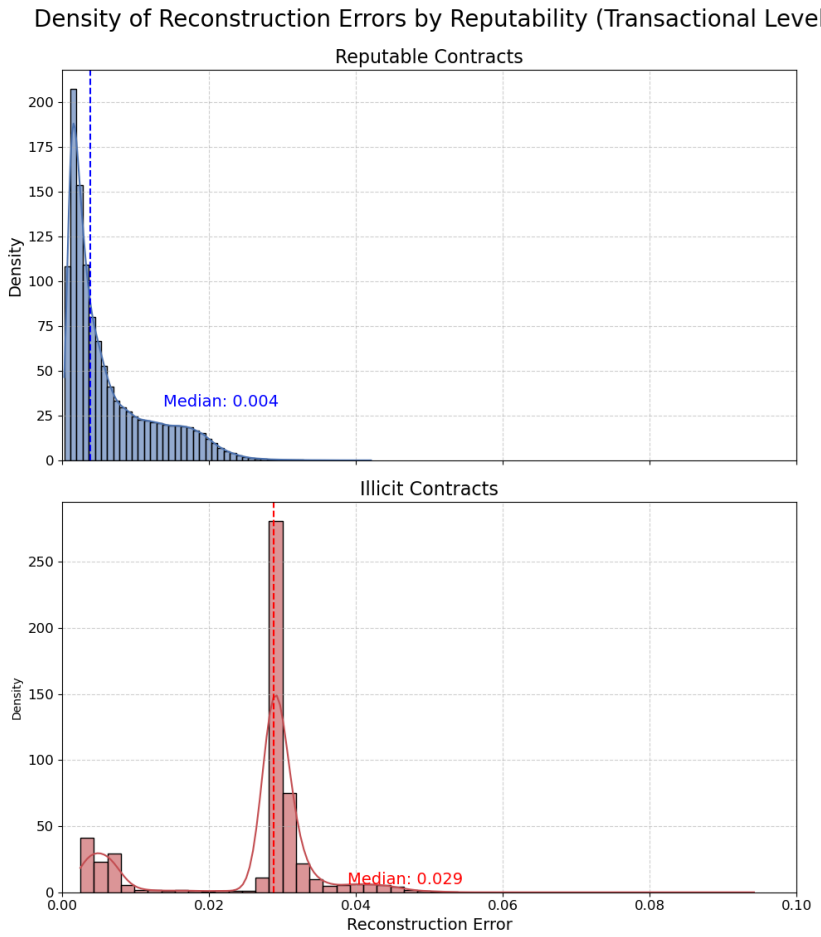


Figure 5.13 Density plots of reconstruction errors from the Multimodal CAE for reputable (blue) and illicit (red) contracts. The autoencoder model generally achieves lower reconstruction errors for reputable contracts.

reconstruction errors for reputable and illicit contracts.

Alternative Hypothesis (H_1): There is a significant difference between the distributions of reconstruction errors for reputable and illicit contracts.

The results of the statistical tests are summarized in Table 5.16.

Test	p-value for Reputable Contracts	p-value for Illicit Contracts	U-statistic / Mann-Whitney U p-value
Shapiro-Wilk Test	4.18×10^{-34}	7.30×10^{-7}	—
Mann-Whitney U Test	—		2.41×10^{-22}
U-Statistic	—		$11,310.0$

Table 5.16 Statistical Test Results for Reconstruction Error Distribution of Reputable and Illicit Contracts

As shown in Table 5.16, the results of the Shapiro-Wilk test indicate that both distributions deviate significantly from normality, as the p-values for reputable and illicit

contract are both less than 0.05, which is our chosen significance level. Given the non-normality of the data, we performed the Mann-Whitney U test, a non-parametric test, to compare the two distributions. The U-statistic was 11,310.0, with a p-value of 2.41×10^{-22}, which is also much smaller than 0.05, indicating a highly significant difference between the two distributions.

These results suggest that, on average, illicit contracts exhibit higher reconstruction errors than reputable ones. This implies that the autoencoder model struggles more to reconstruct illicit contracts, likely due to their anomalous nature.

Finally, we further assess the performance of the Multimodal CAE in classifying reputable and illicit smart contracts, Table 5.17 summarizes the key metrics across various threshold values on the training dataset. Similar to the transaction-only CAE, we identify the optimal threshold using the procedure outlined in Section 4.5.2, and present the results of different performance metrics, including AUC, accuracy, F1-score for both reputable and illicit contracts, and recall for illicit contracts.

Threshold	AUC	Accuracy	F1-Score (Reputable)	F1-Score (Illicit)	Recall (Illicit)
75	0.9566	0.9264	0.9527	0.8344	0.8947
80	0.9550	0.9428	0.9637	0.8645	0.8816
85	0.9699	0.9646	0.9779	0.9116	0.8816
90	**0.9750**	**0.9918**	**0.9949**	**0.9801**	**0.9737**

Table 5.17 Performance metrics across different thresholds for the Multimodal CAE. Metrics include AUC, accuracy, F1-scores for reputable and illicit contracts, and recall for illicit contracts on Training set

As seen in Table 5.17, the multimodal CAE demonstrates consistent performance across various thresholds, aligning with the trend observed in the transaction-only CAE.

A threshold of **90** emerges as the optimal choice, striking a balance between precision and recall. At this threshold, the model achieves an AUC of **0.9750**, accuracy of 99.18%, and an F1-score of 0.9801 for illicit contracts. Moreover, it maintains a high F1-score of 0.9949 for reputable contracts, indicating exceptional overall performance. While the recall for illicit contracts slightly decreases to **97.37%** at this threshold, the significant improvement in accuracy and F1-scores for both reputable and illicit contracts justifies the choice of **90** as the optimal setting.

In conclusion, a threshold of **90** offers the most effective balance between sensitivity and specificity for the multimodal CAE, demonstrating superior performance compared to lower thresholds.

5.2.3 Performance Comparison and Discussion

The second experiment aimed to implement multimodal data integration by combining transaction-level data and opcode embeddings, which directly contributed to Objective 2 of enhancing anomaly detection performance in smart contracts using a multimodal data fusion. By integrating these two data sources, we sought to better understand how combining embedding-based bytecode analysis with transactional behaviour can improve model robustness, particularly for identifying illicit smart contracts. This experiment was crucial for laying the groundwork toward developing a comprehensive framework for reputability prediction and anomaly detection.

The results of Experiment 2 demonstrate that the multimodal autoencoder, which integrates both transaction-level data and opcode embeddings, achieves notable improvements over the transaction-only CAE in several performance metrics. Specifically, the multimodal model yielded higher AUC and F1-scores, particularly for the illicit contracts, which are the most critical class in this study.

The primary advantage of the multimodal approach lies in its ability to leverage both temporal transaction data and bytecode embeddings. The inclusion of opcode embeddings allows the model to better capture patterns in smart contract behaviour that may not be evident from transaction data alone. This enhanced representation is reflected in the improved reconstruction errors for both reputable and illicit contracts, with a significant reduction in median reconstruction error for reputable contracts compared to the transaction-only model. The higher reconstruction errors observed for illicit contracts are particularly beneficial, as they enable more accurate identification of anomalies, thus improving the overall anomaly detection performance.

As illustrated in Figure 5.14, the t-SNE and PCA visualizations show a clearer separation between reputable (blue) and illicit (red) contracts in the multimodal model. The transaction-only autoencoder produces some separation between these two classes, but there is a noticeable overlap. On the other hand, the multimodal autoencoder results in more distinct clustering, particularly for the illicit contracts. This enhanced separation strongly indicates that the integration of opcode embeddings significantly improves the model's ability to differentiate between reputable and illicit contracts by learning more robust latent representations.

Quantitatively, the multimodal CAE consistently outperforms the transaction-only CAE across various thresholds. At the optimal threshold of 90, the multimodal model achieves an AUC of 0.9750 and an F1-score of 0.9801 for illicit contracts, compared to an AUC of 0.9454 and an F1-score of 0.9200 for the transaction-only CAE. The higher F1-score and precision for illicit contracts confirm the superiority of the multimodal approach in detecting anomalies. The performance metrics for both models are summarized in Table 5.18.

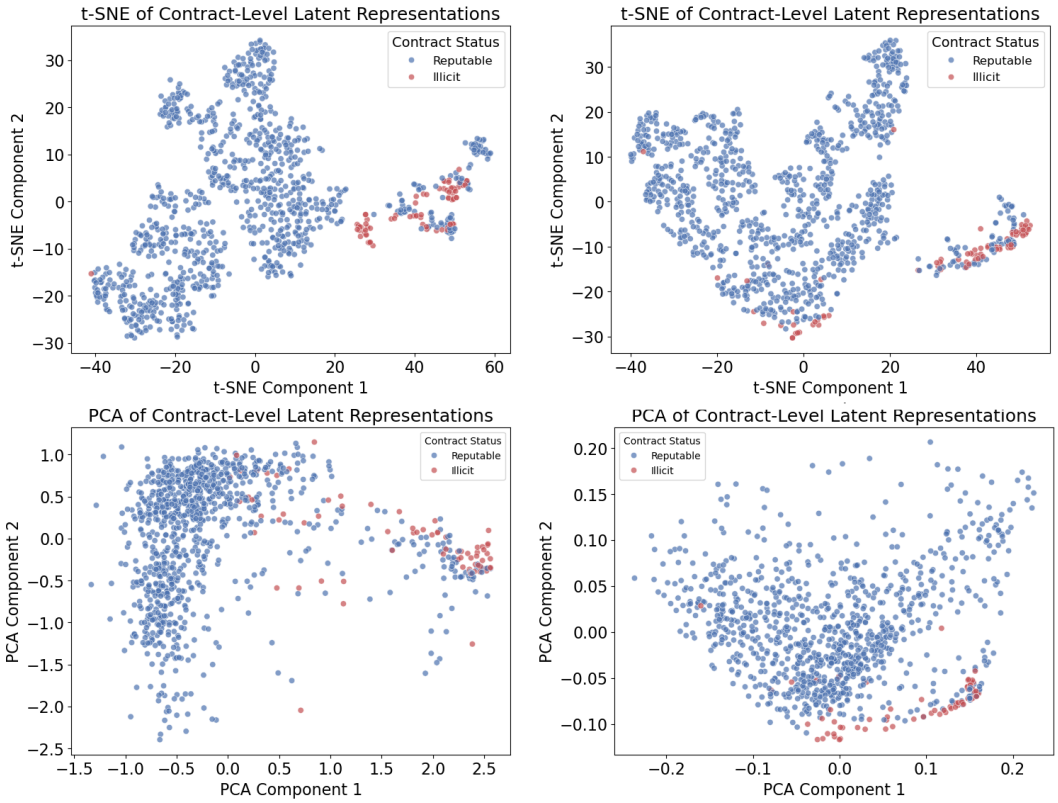


Figure 5.14 t-SNE and PCA plots for unimodal (on the left) and multimodal (on the right) autoencoders

Model	AUC	Accuracy	F1-Score (Reputable)	F1-Score (Illicit)	Recall (Illicit)	Precision (Illicit)
Transaction-Only LSTM-AE	0.9387	95.91	0.9792	0.8861	0.9211	0.9040
Transaction-Only CAE	0.9454	96.73	0.9795	0.9200	0.9079	0.9145
Multimodal CAE	0.9750	99.18	0.9949	0.9801	0.9737	0.9773

Table 5.18 Performance metrics for Transaction-Only LSTM-AE, Transaction-Only CAE, and Multimodal CAE, including AUC, accuracy, F1-scores (reputable and illicit contracts), recall, and precision for illicit contracts on test set.

These results demonstrate the value of integrating opcode embeddings with transaction data. The improved AUC and F1-scores, especially for illicit contracts, highlight that opcode embeddings provide essential context that transaction data alone cannot capture. This confirms that integrating opcode embeddings allows the model to learn complex patterns in smart contract behaviour, contributing to more accurate anomaly detection. The enhanced model's performance metrics such as the 6.53% increase in F1-score and 7.25% improvement in recall for illicit contracts demonstrate that the mul-

timodal approach is a powerful tool for identifying potentially fraudulent contracts with greater precision and recall.

The findings from Experiment 2 are significant for several reasons. First, they validate that the combination of embedding-based bytecode analysis (opcode embeddings) and transactional data offers a more comprehensive understanding of smart contract behaviour. This advancement supports this research's goal of creating a robust framework that can leverage diverse data sources for smarter decision-making in blockchain environments. By achieving higher performance in anomaly detection, the multimodal model provides a practical tool for identifying illicit contracts, which can aid in proactive risk management and enhance the security of blockchain ecosystems.

Moreover, this experiment's success confirms that integrating static and transactional data is not only beneficial but necessary for advancing smart contract analysis. The multimodal model's ability to distinguish between reputable and illicit contracts with clearer boundaries indicates its potential to strengthen security measures. This result sets the stage for subsequent phases of the study, where temporal data will be integrated to analyse the evolution of reputability over time. It also suggests that future models should adopt multimodal approaches as a standard practice for anomaly detection and reputability assessment, given their significant improvements over unimodal methods. By leveraging complementary insights from multiple data sources, these approaches enhance detection accuracy and reliability.

The conclusions drawn from Experiment 2 go beyond confirming that multimodal integration improves model performance. They signify that leveraging diverse data modalities can enhance the depth and accuracy of predictive frameworks, making them more capable of adapting to the complex nature of smart contracts. This finding implies that for a robust and comprehensive approach to smart contract analysis, incorporating diverse data sources is essential.

In summary, through Experiment 2, we successfully achieved Objective 2 i.e., demonstrating the benefits of combining opcode embeddings with transaction-level data in anomaly detection. This not only strengthens the validity of the proposed multimodal framework but also establishes a foundation for the next phase of the study, where temporal analysis will be incorporated. The integration of these findings into the overall thesis aligns with the aim of developing a framework that enables real-time, dynamic evaluations of smart contract reputability and anomaly detection, contributing to a more secure and reliable blockchain environment.

5.3 Experiment 3: Long-Term Temporal Analysis of Smart Contract Reputability

This experiment aims to evaluate the performance of two deep learning models, LSTM and CNN-LSTM with Multihead Attention, in predicting long-term trends in smart contract reputability scores hence addressing Objective 3 (Section 1.3). Both models are designed to handle time-series data, with the LSTM focusing on temporal dependencies and the CNN-LSTM model leveraging CNN layers and an attention mechanism to capture both local patterns and significant time-based trends. The goal is to identify the model that best captures reputability trends, maintaining accuracy and smoothness in predictions over extended periods.

LSTM Units	Batch Size	Learning Rate	Avg MAE	Avg Val Loss	Best Epoch (Avg)
128	32	0.0005	0.1586	0.0429	14
64	32	0.001	0.1591	0.0430	12
128	64	0.0005	0.1587	0.0432	16
128	64	0.001	0.1598	0.0431	15
32	64	0.0005	0.1602	0.0434	11

Table 5.19 Top 5 LSTM Model Configurations by Average Validation Loss Across Folds

The results in Table 5.19 show the top-performing LSTM configurations based on validation loss across folds. The configuration with 128 LSTM units, a batch size of 32, and a learning rate of 0.0005 achieved the lowest average validation loss (0.0429) and maintained low MAE, indicating strong accuracy in trend prediction. This aligns with the findings in [79], who noted the LSTM's effectiveness in capturing long-term dependencies in sequential data. The relatively low error values across configurations suggest that the LSTM model effectively captures reputability trends while minimizing overfitting to minor fluctuations.

Num Filters	LSTM Units	Learning Rate	Dropout Rate	Avg Val Loss	Avg MAE
64	32	0.0005	0.1	0.0443	0.1623
128	64	0.0001	0.1	0.0452	0.1649
128	32	0.001	0.3	0.0454	0.1658
32	32	0.001	0.1	0.0454	0.1635
128	32	0.0005	0.3	0.0456	0.1661

Table 5.20 Top 5 CNN-LSTM with Multihead Attention Configurations by Average Validation Loss Across Folds

Table 5.20 presents the top configurations for the CNN-LSTM with Multihead Attention model. The configuration with 64 filters, 32 LSTM units, a learning rate of

0.0005, and a dropout rate of 0.1 achieved the lowest average validation loss (0.0443). The attention mechanism in this model [151] allows it to focus on critical segments within the sequence, which enhances its performance on data with complex patterns and shifts. However, the CNN-LSTM model's slightly higher MAE suggests that the added complexity of CNN and attention layers may lead to sensitivity to local variations, consistent with the findings in [152].

To further assess the stability and robustness of each model, we examined the training and validation loss across epochs and averaged across folds. The plots in Figure 5.15 depict the average training and validation loss per epoch for the LSTM and CNN-LSTM models, respectively.

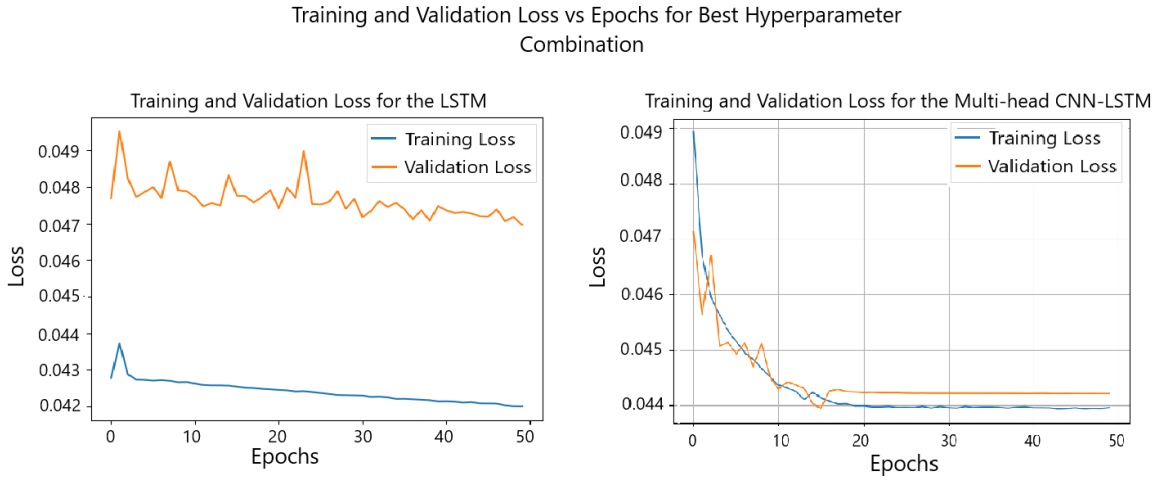


Figure 5.15 Average Training and Validation Loss Across Folds for the LSTM (left) and CNN-LSTM (right) models. The LSTM model shows stable convergence with minimal variance, while the CNN-LSTM model exhibits slight fluctuations in validation loss, indicating higher sensitivity to local variations.

The LSTM model demonstrates consistent convergence with minimal variance across folds, maintaining a low validation loss, reinforcing its suitability for long-term trend monitoring. the validation loss of CNN-LSTM with Multihead Attention exhibits slight fluctuations, suggesting higher sensitivity to local data variations. This observation aligns with our earlier analysis, indicating that while the CNN-LSTM model benefits from the added attention mechanism, it may be prone to overfitting to short-term changes.

In comparing the two models, the LSTM outperformed the CNN-LSTM with Multihead Attention in terms of lower validation loss and MAE, indicating a superior ability to capture stable, long-term trends in reputability scores. The simpler architecture of the LSTM model, without convolutional or attention layers, proves advantageous in this task by avoiding overfitting to minor fluctuations.

For further details on individual validation losses across folds and configurations, see Appendix E.

5.3.1 Qualitative Analysis

Since the LSTM model emerges as the more reliable choice for this experiment with marginal better performance than the CNN-LSTM model, we place our primary focus on this model for further assessment.

To further assess model performance, a qualitative analysis was conducted to evaluate the LSTM model's effectiveness in capturing long-term trends in reputability scores, as discussed in the methodology. This analysis begins with examining the distribution of reputability scores among reputable and illicit smart contracts, providing insights into the model's ability to differentiate these classes effectively.

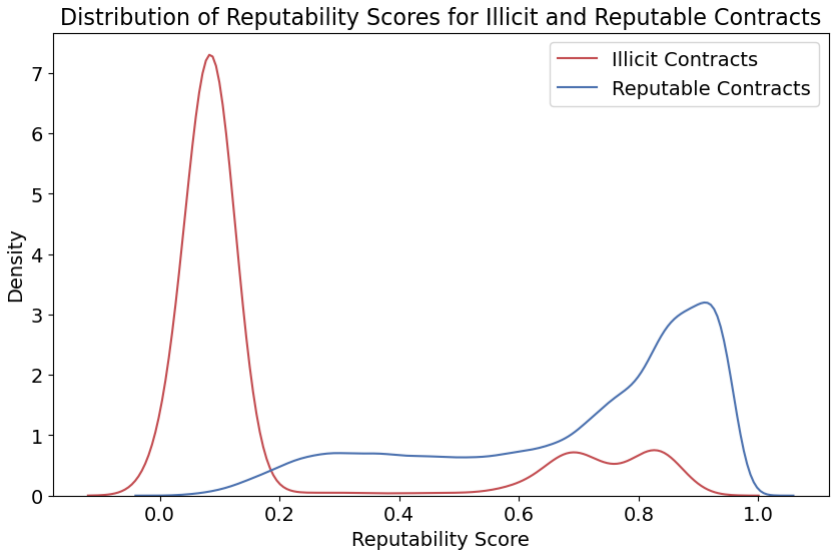


Figure 5.16 Distribution of Reputability Scores for Illicit and Reputable Contracts

As shown in Figure 5.16, the distribution of scores highlights a clear distinction between reputable and illicit contracts. Reputable contracts tend to have higher reputability scores, with a peak around 0.8 to 1.0, indicating consistent classification as reputable. In contrast, illicit contracts predominantly have scores near 0.1 to 0.2, suggesting that they are consistently recognized as illicit. This separation supports the hypothesis that the model can maintain stability in reputability scores across classes, with reputable contracts remaining within the high range and illicit contracts clustering within the low range.

To further investigate the model's effectiveness in tracking diverse reputability trends, we examined four distinct smart contracts, each representing a unique trend type: Increasing, Decreasing, Stable, and High-Variance. These samples allow us to visually assess the LSTM model's capability to capture various reputability patterns over time.

In Figure 5.17, the performance of the LSTM model on each sample contract is illustrated, highlighting the model's ability to adapt to different trend types:

- **Increasing Trend**: For contracts exhibiting an upward trend in reputability, the LSTM model captures the gradual rise effectively, with predictions closely matching the actual scores. This demonstrates the model's robustness in recognizing contracts that are becoming progressively more reputable over time.
- **Decreasing Trend**: In cases where contracts show a downward trend in reputability, the model accurately reflects the decline, with predictions aligning closely with the observed scores. This capability is essential for detecting reputability degradation and tracking contracts that may be losing their reputable status.
- **Stable Trend**: For contracts with stable reputability scores, the LSTM model maintains a consistent prediction line, effectively resisting overfitting to minor fluctuations. This stability indicates the model's ability to represent contracts with minimal score variance, where reputability remains largely constant.

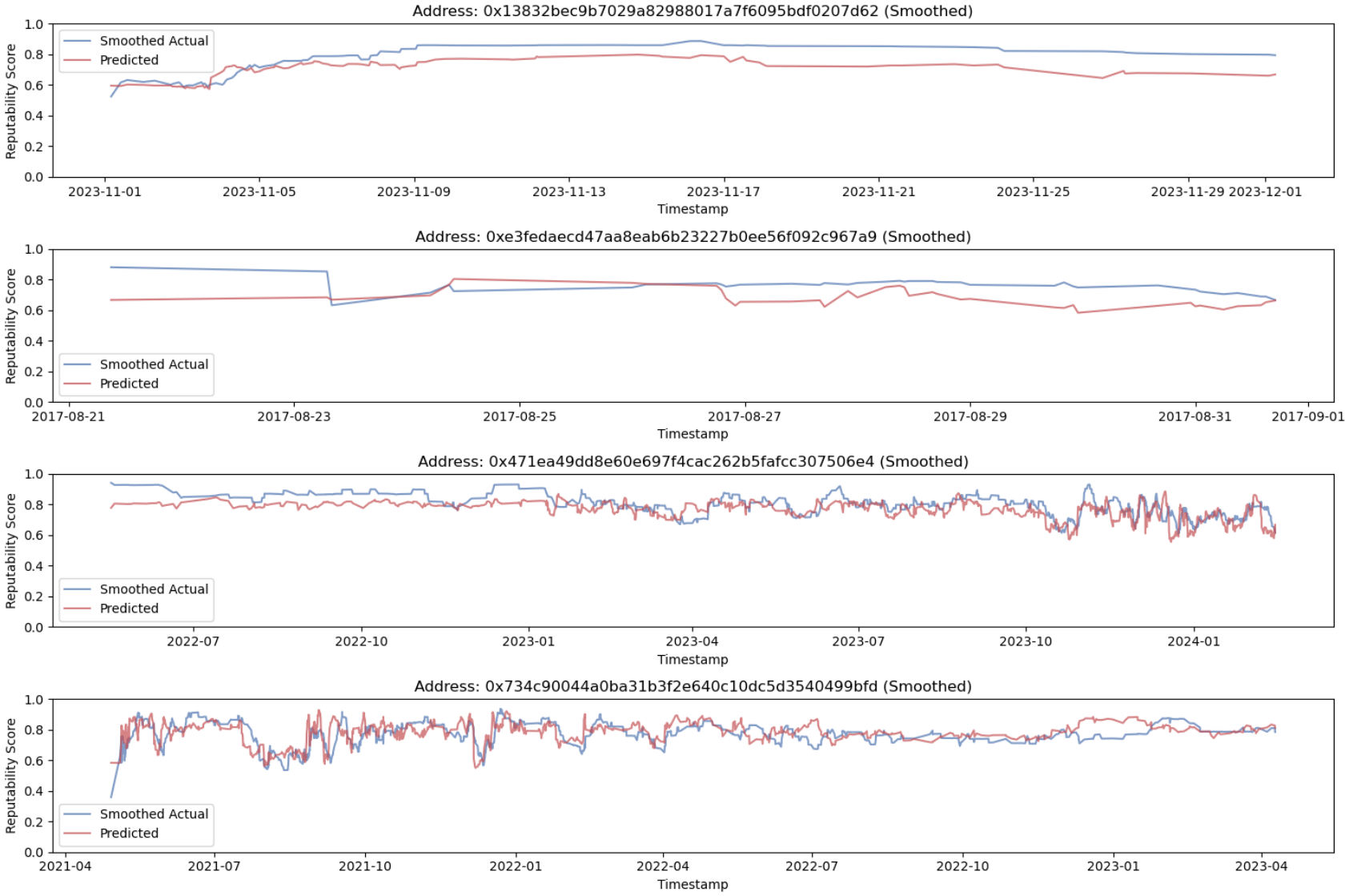


Figure 5.17 LSTM Model Output for Reputability Scores on Selected Smart Contracts with Different Trend Patterns

- **High-Variance Trend**: For contracts with high score variability, where reputability scores fluctuate significantly, the LSTM model provides a smoothed approximation of the trend, capturing the overall direction without reacting to every short-term change. This smoothing approach is beneficial for applications prioritizing long-term stability, though it may overlook minor shifts in highly volatile contracts.

Overall, these qualitative insights illustrate the LSTM model's strengths in maintaining long-term trend accuracy across different scenarios. The ability to differentiate between reputable and illicit contracts, as well as to adapt to various trend patterns, supports the model's utility in reputability analysis for smart contracts.

Next, to assess prediction lag, a cross-correlation analysis and MSE calculations at various lags were conducted.

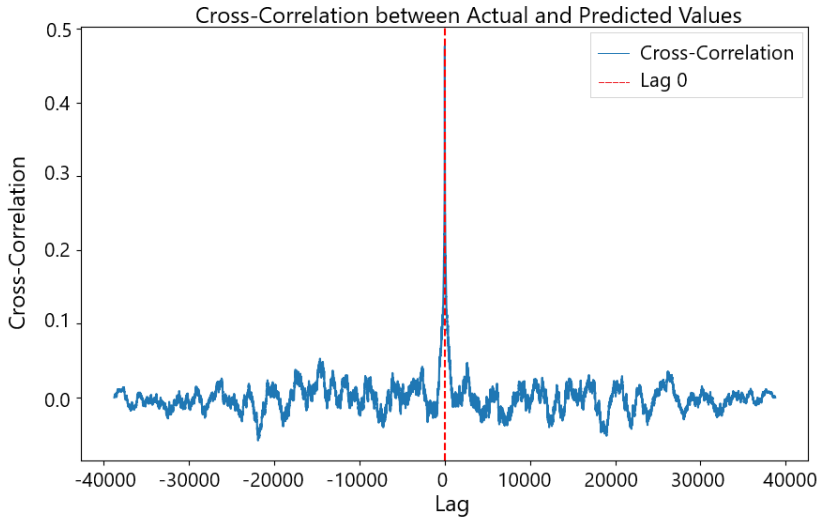


Figure 5.18 Cross-Correlation between Actual and Predicted Reputability Scores

As shown in Figure 5.18, the cross-correlation plot exhibits a prominent peak at *Lag 0*, indicating strong alignment between the actual and predicted values with minimal lag. This suggests that the model's predictions are well-synchronized with the actual trend.

Additionally, MSE values were calculated to quantify prediction lag across small shifts. The MSE at *Lag 0* was 0.0463, with minor increases at *Lag 1* (0.0467) and *Lag 2* (0.0471). These results confirm that introducing a lag does not significantly reduce prediction error, further supporting that the model's predictions align closely with actual values without noticeable delay.

Finally, we examine the rolling window error plot (Figure 5.19) to evaluate the model's temporal consistency in capturing reputability trends. The plot displays the rolling MSE and MAE over time, with a window size of 25. The consistent, low values for the rolling MSE (in blue) indicate that the LSTM model maintains stable performance

throughout the time series, without significant fluctuations in error. This stability aligns with our objective of using LSTM models for long-term reputability analysis, as it demonstrates the model's robustness in tracking broader trends rather than reacting to minor variations.

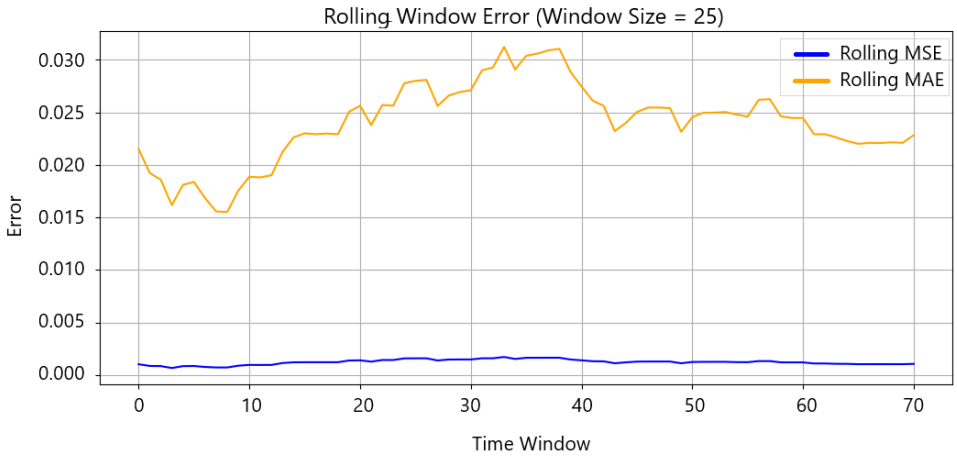


Figure 5.19 Rolling Window Error Plot (Window Size = 25) for LSTM Model: Rolling MSE and MAE over time, showing the model's consistency in capturing long-term reputability trends.

On the other hand, the rolling MAE (in orange) shows a gradual increase in certain segments, suggesting periods where absolute errors grow, potentially due to increased variability or shifts in reputability that the model finds challenging to capture precisely. This divergence between MSE and MAE may indicate that while the model is generally aligned with the long-term trend, it occasionally exhibits greater deviation in absolute terms.

5.3.2 Model Insights and Discussion

This experiment underscores the potential of LSTM models for monitoring reputability trends in smart contracts over extended periods, emphasizing the unique strengths of LSTMs in tracking and understanding long-term behaviour thereby addressing Objective 3 (Section 1.3). Starting with both LSTM and CNN-LSTM architectures, we found that the LSTM model outperformed the more complex CNN-LSTM model in capturing sustained reputability patterns, achieving a low validation loss of 0.04. This result suggests that simpler architectures, like LSTM, can be highly effective in use cases where stability and consistency are more critical than immediate responsiveness to minor variations.

One of the most notable insights from this study is that the LSTM model not only tracks reputability with high accuracy but also resists overfitting to short-term noise. By focusing on the broader trajectory rather than small, temporary fluctuations, the LSTM

model delivers a more reliable measure of reputability over time. This is particularly valuable in blockchain contexts, where smart contract reputability often changes gradually rather than abruptly. Therefore, the LSTM's ability to "smooth out" short-term anomalies makes it ideal for continuous monitoring, enabling stakeholders to make informed decisions based on stable reputability trends.

The ability to accurately track and predict long-term reputability trends in smart contracts has significant implications for blockchain security and risk management. The consistent performance of the LSTM model across various metrics and its ability to differentiate between stable and high-variance trends validate that the model can be effectively utilized for ongoing reputability analysis. This provides blockchain platforms, developers, and auditors with a tool for continuous assessment, identifying potential risks before they escalate.

In examining individual contract reputability trajectories, certain patterns become apparent that can reveal risks before they materialize. For instance, the contract *0x471ea49dd8e60e697f4cac262b5fafcc307506e4* [1] appeared stable in reputability for most of its operational life. However, around October 2023, the LSTM model detected a significant drop in the reputability score. Upon further examination of the contract code, it was found that the contract's owner had the authority to modify token balances at other addresses. This capability raises a potential risk for asset loss, as it grants the owner discretionary control over user balances. Such insights underscore the model's utility in preemptively identifying reputability risks associated with potentially problematic code functionalities, allowing for timely interventions and risk mitigation.

The rolling window error analysis showed that the model's errors remained consistently low across time windows, reinforcing its ability to maintain predictive accuracy without significant drift. This consistency is essential in real-world applications, as it indicates that the model's performance does not degrade over time, offering sustained insights that align with the gradual evolution of reputability in most smart contracts. The low prediction lag further validates that LSTM predictions are well-aligned with actual trends, capturing reputability shifts with minimal delay.

The results also revealed that integrating domain-specific knowledge, such as the identification of code vulnerabilities and patterns of abnormal transaction behaviour, significantly enhanced the interpretability of LSTM model outputs. By correlating sudden reputability declines with specific contract features or transactional anomalies, stakeholders can gain actionable insights into the underlying causes of reputability shifts. This alignment between model predictions and real-world observations underscores the practicality and reliability of LSTM-based monitoring systems for proactive risk management. By bridging predictive insights to blockchain activities, these systems enhance detection of reputability shifts and empower stakeholders to respond effectively to risks.

[1] `https://etherscan.io/address/0x471ea49dd8e60e697f4cac262b5fafcc307506e4`

In conclusion, the findings from Experiment 3 affirm that LSTM models are well-suited for detecting long-term trends in the reputability of smart contracts, offering a more reliable and insightful analysis than short-term anomaly detection methods. This achievement meets Objective 3 of the research and highlights the LSTM model's potential as a foundational tool for continuous, long-term monitoring of contract health. The consistently low validation loss and the stability demonstrated in rolling error metrics validate the model's effectiveness over extended periods. These characteristics position LSTM models as a valuable asset for blockchain platforms, regulatory bodies, and developers, enhancing the ability to track reputability shifts with minimal lag and stable predictive performance. This capability is essential for proactive risk management, enabling timely interventions to maintain the integrity of smart contracts and supporting sustainable, risk-aware blockchain ecosystems.

5.4 Summary

Experiment 1 successfully achieved the objective of evaluating the performance of boosting algorithms in predicting the reputability of smart contracts based on embedding-based bytecode analysis. By leveraging GAN-augmented opcode embeddings, the LightGBM model outperformed other algorithms, including XGBoost and CatBoost, with an accuracy of 97.67% and a recall of 0.942 for illicit contracts. The use of data augmentation through GANs effectively addressed class imbalance, showcasing its superiority over traditional techniques such as SMOTE and ADASYN. This high precision and recall demonstrated the model's capacity to accurately identify potentially illicit contracts, providing a solid base for future research that seeks to integrate bytecode embeddings with other data types.

Building upon the success of Experiment 1, Experiment 2 aimed to enhance the predictive capabilities of smart contract reputability by integrating embedding-based bytecode analysis embeddings with transactional data through a multimodal data fusion framework. The results confirmed the effectiveness of this approach, as the multimodal integration yielded a 7.25% improvement in recall and demonstrated more robust performance across various validation sets compared to single-source models. This experiment underscored that the combination of embedding-based bytecode analysis and transaction history offers a more comprehensive understanding of smart contract behaviour, thus improving the predictive accuracy and reliability of the models. The success of Experiment 2 in incorporating both static and transactional data laid the groundwork for Experiment 3, which focused on long-term monitoring and trend analysis, advancing the ability to capture reputability shifts over time.

Leveraging the framework established in Experiment 2, Experiment 3 explored

the use of LSTM models for long-term monitoring of smart contract reputability, focusing on identifying stable, evolving trends. The findings confirmed that LSTM models were effective, achieving a validation loss of 0.0429 and maintaining consistently low MAE across different time windows. The model's strong performance in rolling window error analyses reinforced its ability to maintain predictive stability and detect reputability shifts with minimal lag. This demonstrated that LSTM models outperform more complex architectures like CNN-LSTM for long-term trend analysis due to their focus on sequential dependencies without overreacting to short-term fluctuations. The sustained accuracy and minimal prediction lag highlight LSTM models' potential for blockchain monitoring, contributing a valuable tool for early detection and proactive risk management. This final experiment underscores the cumulative impact of integrating bytecode embeddings, transactional data, and sequential modelling techniques for comprehensive smart contract reputability assessment.

6 Conclusion

This research was motivated by the need for more reliable reputability metrics in blockchain applications, where the risk of vulnerabilities and malicious activities in smart contracts poses a critical challenge. This chapter revisits each objective and reflects on our achievements and whether the research aim was fully realized.

6.1 Summary of Objectives and Achievements

In this study, we proposed a multimodal data fusion framework for enhancing the analysis of smart contract reputability on the Ethereum blockchain. The primary aim was to integrate embedding-based bytecode analysis with transactional data, providing a comprehensive approach to reputability assessment.

O1: Acquire, preprocess, and analyse a balanced dataset of smart contracts to evaluate the performance of boosting algorithms in predicting reputability based on bytecode embeddings.

The first objective was to develop a predictive framework for assessing smart contract reputability based on bytecode embeddings, using boosting algorithms to address the complexities of imbalanced datasets. Through the application of data augmentation techniques like SMOTE, ADASYN, and GANs, we effectively balanced the dataset, enhancing the model's ability to generalize across different contract types. Our LightGBM model demonstrated remarkable performance on the benchmark dataset, achieving better results than the original study [97], which underscores the strength of this approach in reputability prediction. By leveraging bytecode embeddings, we captured essential code-level attributes that differentiate reputable and illicit contracts, achieving high recall and F1 scores, particularly for the minority (illicit) class. These findings affirm the value of embedding-based bytecode analysis in reputability assessment and establish LightGBM as a robust tool for identifying potentially risky contracts.

The superior performance of the LightGBM model on this benchmark highlights the critical role of static features in reputability analysis, particularly when combined with targeted data augmentation. This approach not only enhances model accuracy but also provides a scalable, code-based assessment method for early risk detection, enabling stakeholders to make informed decisions without requiring extensive transactional history. By focusing on the static aspects of smart contracts, we offer a proactive approach that can be readily integrated into smart contract vetting processes, reinforcing blockchain security from a code-centric perspective. Furthermore, this method ensures that potential vulnerabilities can be identified before a contract is deployed, reducing the risk of costly exploits and fostering greater confidence in the reliability of smart con-

tract ecosystems. This proactive strategy is especially beneficial in environments where rapid contract deployment is common, allowing for swift evaluation and mitigation of risks without disrupting development timelines.

O2: Implement multimodal data fusion by integrating bytecode embeddings with transactional data, and apply anomaly detection techniques to detect abnormal patterns in smart contracts over time

Achieving the second objective involved developing an anomaly detection framework that combines bytecode embeddings with transactional data, thereby leveraging a multimodal approach to capture a more comprehensive view of reputability. Using a CNN-based autoencoder for transaction-level anomaly detection, we successfully detected reputability shifts by integrating these dynamic insights with bytecode embeddings. The multimodal fusion of static and transactional data resulted in significant improvements over single-modality models, providing enhanced accuracy and recall in anomaly detection. This supports our hypothesis that integrating multiple data types offers a more nuanced understanding of reputability, as static and dynamic features together reveal patterns that neither modality could fully capture alone.

These results underscore the practical benefits of multimodal fusion for smart contract reputability analysis. By capturing both inherent code vulnerabilities and evolving transaction patterns, this model provides a robust foundation for continuous reputability monitoring, enabling early intervention when anomalies arise. The added accuracy and recall achieved with multimodal analysis are especially valuable in high-stakes environments where a single missed anomaly could result in considerable financial or reputational losses. This approach represents a promising direction for enhancing blockchain trustworthiness, as it enables a balanced assessment of both static and behavioural characteristics of smart contracts.

O3: Predict how the reputability and anomaly score evolve over time by analysing temporal transaction data using sequence modelling technique

The final objective centred on predicting the evolution of reputability and anomaly scores over time, specifically focusing on long-term reputability patterns that offer critical insights for monitoring smart contract behaviour. By implementing both LSTM and CNN-LSTM with Multi-head Attention models, we explored methods to capture stable, time-based trends in reputability. The LSTM model, with its focus on capturing sequential dependencies, outperformed the CNN-LSTM model, achieving consistently lower validation loss and demonstrating a robust ability to model reputability trajectories without overfitting to short-term fluctuations. This reliability in long-term trend analysis enables a stable foundation for ongoing reputability assessments, enabling monitoring of gradual reputability changes with confidence.

The findings from this experiment highlight the value of analysing long-term reputability patterns in smart contract monitoring and risk management. The LSTM model's

ability to smooth out transient noise and focus on overarching reputability shifts proves especially useful for early detection of reputability declines, which can serve as early warning signals for potential vulnerabilities. This capacity for tracking reputability evolution over time allows stakeholders to proactively identify risks, making it possible to mitigate issues before they impact the broader blockchain ecosystem. Consequently, the integration of such long-term temporal models into smart contract monitoring systems offers a strategic advantage, fostering a more secure blockchain environment.

6.2 Revisiting the Research Aim

The overarching aim of this research was to determine whether a multimodal data fusion framework could effectively predict the evolution of smart contract reputability over time by integrating embedding-based bytecode analysis and dynamic transactional data, even in the absence of explicit reputability labels at specific timestamps. This question was motivated by the need for a more comprehensive reputability assessment model in blockchain systems, where traditional approaches often rely solely on code analysis or transactional patterns in isolation. By fusing these two data types, we sought to establish a robust and comprehensive model capable of capturing both inherent code vulnerabilities and evolving transactional behaviours.

Since all the objectives outlined in Section 4.1 have been successfully met, we have achieved our research aim. The LightGBM model, through embedding-based bytecode analysis and data augmentation, demonstrated high accuracy in predicting reputability based on static features alone. Additionally, the multimodal anomaly detection model provided an in-depth understanding of reputability trends by capturing both static and dynamic data perspectives. Finally, the temporal sequence analysis in Objective 3 further extended our framework's ability to track reputability evolution, confirming that this multimodal approach effectively supports reputability prediction over time, regardless of the presence of explicit reputability labels. Together, these outcomes validate the proposed framework's capacity to provide a reliable, holistic reputability assessment for smart contracts, addressing a crucial gap in blockchain security research.

6.3 Limitations

While this study achieved its main objectives and provided valuable insights, several limitations were identified that could be addressed in future work to further enhance the robustness and applicability of the research.

1. Data Imbalance
   One major challenge encountered was the data imbalance between reputable and illicit smart contracts. There was an abundance of reputable accounts with many transactions, leading to a large volume of reputable data compared to the limited availability of illicit contracts. Additionally, reputable accounts tended to have more transactions on average than illicit ones, compounding the imbalance. Although we used data augmentation techniques to mitigate this, the limited diversity and quantity of illicit contract data may have constrained the model's ability to generalize fully, as synthetic data often struggles to capture the full range of real-world patterns in illicit contracts.

2. Label Ambiguity in Reputability
   Since explicit reputability labels at specific timestamps were unavailable, we inferred reputability trends from proxy indicators like reconstruction errors or reputability scores derived from model predictions. This approach introduces potential ambiguity, as inferred scores may not always align with real-world reputability shifts. This lack of precise labelling can impact the model's interpretability and the robustness of predictions, especially in edge cases where reputability changes are subtle or irregular.

3. Complexity of Multimodal Fusion
   The multimodal fusion approach increased the complexity of the model, both computationally and in terms of implementation. The integration of static and transactional features required careful tuning, as discrepancies in data modalities led to challenges in achieving an optimal balance. Additionally, computational constraints limited the potential for hyperparameter tuning on a larger scale, potentially impacting the full optimization of the multimodal model.

6.4 Future Work

Building on the findings of this study, several promising directions have been identified for future research to enhance smart contract reputability prediction and monitoring. One potential advancement lies in the incorporation of real-time data streams to improve the model's temporal accuracy and responsiveness. By using high-frequency data, the model could capture finer-grained trends in reputability, enabling more immediate detection of reputability shifts as they occur. This real-time responsiveness would be especially valuable in blockchain applications where rapid reaction to reputability changes is crucial for minimizing potential risks and losses. Moreover, such capabilities can help prevent vulnerabilities from escalating.

Another avenue for improvement is the exploration of advanced data fusion techniques to strengthen the interactions between embedding-based bytecode analysis and transactional features. In particular, attention mechanisms could be employed to dynamically prioritize features based on their relevance at each point in the sequence. Such mechanisms could significantly enhance the model's ability to capture nuanced patterns within the data, as attention layers allow the model to focus on the most influential features for reputability prediction. This refined approach to multimodal fusion could lead to improved model performance, especially in scenarios where relationships between static and transactional data are complex and situational.

Finally expanding the dataset, especially by including more diverse illicit smart contracts, could enhance the model's robustness and generalizability. Acquiring additional data from blockchain monitoring platforms or through industry collaborations would provide a broader range of behavioural patterns and reputability dynamics. A more balanced and representative dataset would enable better model training and evaluation, mitigating the constraints of current data limitations. These improvements would contribute to a more accurate, responsive, and widely applicable framework for smart contract reputability analysis.

6.5 Concluding Remarks

This research has demonstrated the potential of a multimodal data fusion framework to enhance smart contract reputability analysis. By integrating embedding-based bytecode analysis with dynamic transactional data and employing advanced techniques in anomaly detection and temporal sequence modelling, we developed a comprehensive framework that addresses significant challenges in blockchain security. Through fulfilling each objective—effective code-based reputability prediction, multimodal anomaly detection, and long-term reputability trend analysis—this study validated that a combined approach can provide a more accurate and holistic view of reputability trends, even in the absence of explicit reputability labels.

Although limitations were encountered, including data imbalance, label ambiguity, and the complexities of multimodal fusion, the outcomes affirm that combining static and dynamic data types creates a powerful framework for reputability assessment. Future work that expands on these insights—such as real-time data integration and more sophisticated fusion techniques—has the potential to further enhance blockchain reliability, fostering greater trust in decentralized applications. This study represents a significant step toward understanding and predicting smart contract reputability, providing foundational insights for both future research and practical applications in the rapidly evolving blockchain landscape.

References

[1] M. Kolvart, M. Poola, and A. Rull, "Smart contracts," *The Future of Law and etechnologies*, pp. 133–147, 2016.

[2] N. Szabo, "Formalizing and securing relationships on public networks," *First monday*, 1997.

[3] S. Sayeed, H. Marco-Gisbert, and T. Caira, "Smart contract: Attacks and protections," *Ieee Access*, vol. 8, pp. 24 416–24 427, 2020.

[4] A. Groce, J. Feist, G. Grieco, and M. Colburn, "What are the actual flaws in important smart contracts (and how can we find them)?" In *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*, Springer, 2020, pp. 634–653.

[5] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "Smartcheck: Static analysis of ethereum smart contracts," in *Proceedings of the 1st international workshop on emerging trends in software engineering for blockchain*, 2018, pp. 9–16.

[6] A. M. Rozario and M. A. Vasarhelyi, "Auditing with smart contracts.," *International Journal of Digital Accounting Research*, vol. 18, 2018.

[7] P. Qian, Z. Liu, Q. He, B. Huang, D. Tian, and X. Wang, "Smart contract vulnerability detection technique: A survey," *arXiv preprint arXiv:2209.05872*, 2022.

[8] S. Farrugia, J. Ellul, and G. Azzopardi, "Detection of illicit accounts over the ethereum blockchain," *Expert Systems with Applications*, vol. 150, p. 113 318, 2020.

[9] R. F. Ibrahim, A. M. Elian, and M. Ababneh, "Illicit account detection in the ethereum blockchain using machine learning," in *2021 international conference on information technology (ICIT)*, IEEE, 2021, pp. 488–493.

[10] C. Obi-Okoli, O. Jogunola, B. Adebisi, and M. Hammoudeh, "Machine learning algorithms to detect illicit accounts on ethereum blockchain," in *Proceedings of the 7th International Conference on Future Networks and Distributed Systems*, 2023, pp. 747–752.

[11] C. Malik, J. Ellul, and J. Bajada, *Identifying likely reputable ethereum blockchain based projects*, Bachelor's Thesis, Unpublished, Msida, Malta, 2023.

[12] M. Staples *et al.*, "Risks and opportunities for systems using blockchain and smart contracts. data61," *CSIRO), Sydney*, 2017.

[13] C.-Y. Lin, H.-K. Liao, and F.-C. Tsai, "A systematic review of detecting illicit bitcoin transactions," *Procedia Computer Science*, vol. 207, pp. 3217–3225, 2022.

[14] C. Cholevas, E. Angeli, Z. Sereti, E. Mavrikos, and G. E. Tsekouras, "Anomaly detection in blockchain networks using unsupervised learning: A survey," *Algorithms*, vol. 17, no. 5, p. 201, 2024.

[15] D. He, Z. Deng, Y. Zhang, S. Chan, Y. Cheng, and N. Guizani, "Smart contract vulnerability analysis and security audit," *IEEE Network*, vol. 34, no. 5, pp. 276–282, 2020.

[16] L. Zhao, S. Sen Gupta, A. Khan, and R. Luo, "Temporal analysis of the entire ethereum blockchain network," in *Proceedings of the Web Conference 2021*, 2021, pp. 2258–2269.

[17] R. Agarwal, S. Barve, and S. K. Shukla, "Detecting malicious accounts in permissionless blockchains using temporal graph properties," *Applied Network Science*, vol. 6, pp. 1–30, 2021.

[18] D. Freeman, T. McWilliams, S. Bhattacharyya, C. Hall, and P. Peillard, "Enhancing trust in the cryptocurrency marketplace: A reputation scoring approach," *SMU Data Science Review*, vol. 1, no. 3, p. 5, 2018.

[19] Y. Yuan and F.-Y. Wang, "Blockchain and cryptocurrencies: Model, techniques, and applications," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 9, pp. 1421–1428, 2018.

[20] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system bitcoin: A peer-to-peer electronic cash system," *Bitcoin. org. Disponible en https://bitcoin. org/en/bitcoin-paper*, 2009.

[21] S. S. Sarmah, "Understanding blockchain technology," *Computer Science and Engineering*, vol. 8, no. 2, pp. 23–29, 2018.

[22] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," *self-published paper, August*, vol. 19, no. 1, 2012.

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

[24] L. Vishwakarma and D. Das, "Bss: Blockchain enabled security system for internet of things applications," in *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*, IEEE, 2020, pp. 1–4.

[25] V. Buterin, "Ethereum: Platform review," *Opportunities and challenges for private and consortium blockchains*, vol. 45, pp. 1–45, 2016.

[26] M. I. Mehar *et al.*, "Understanding a revolutionary and flawed grand experiment in blockchain: The dao attack," *Journal of Cases on Information Technology (JCIT)*, vol. 21, no. 1, pp. 19–32, 2019.

[27] K. Christidis and M. Devetsikiotis, “Blockchains and smart contracts for the internet of things,” *IEEE access*, vol. 4, pp. 2292–2303, 2016.

[28] Q. Lu *et al.*, “Integrated model-driven engineering of blockchain applications for business processes and asset management,” *Software: Practice and Experience*, vol. 51, no. 5, pp. 1059–1079, 2021.

[29] A. Turner and A. S. M. Irwin, “Bitcoin transactions: A digital discovery of illicit activity on the blockchain,” *Journal of Financial Crime*, vol. 25, no. 1, pp. 109–130, 2018.

[30] I. Alarab, S. Prakoonwit, and M. I. Nacer, “Competence of graph convolutional networks for anti-money laundering in bitcoin blockchain,” in *Proceedings of the 2020 5th international conference on machine learning technologies*, 2020, pp. 23–27.

[31] A. Gucciardi, “Trustless contract management: A study on the benefits of blockchain-based smart contracts,” Ph.D. dissertation, Politecnico di Torino, 2023.

[32] J. Feist, G. Grieco, and A. Groce, “Slither: A static analysis framework for smart contracts,” in *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, IEEE, 2019, pp. 8–15.

[33] F. Hofmann, S. Wurster, E. Ron, and M. Böhmecke-Schwafert, “The immutability concept of blockchains and benefits of early standardization,” in *2017 ITU Kaleidoscope: Challenges for a Data-Driven Society (ITU K)*, IEEE, 2017, pp. 1–8.

[34] D. Perez and B. Livshits, “Smart contract vulnerabilities: Does anyone care,” *arXiv preprint arXiv:1902.06710*, pp. 1–15, 2019.

[35] G. Grieco, W. Song, A. Cygan, J. Feist, and A. Groce, “Echidna: Effective, usable, and fast fuzzing for smart contracts,” in *Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis*, 2020, pp. 557–560.

[36] P. Gill, I. Ray, A. L. Takami, and M. Tripunitara, “Finding unchecked low-level calls with zero false positives and negatives in ethereum smart contracts,” in *International Symposium on Foundations and Practice of Security*, Springer, 2022, pp. 305–321.

[37] F. A. Alaba, H. A. Sulaimon, M. I. Marisa, and O. Najeem, “Smart contracts security application and challenges: A review,” *Cloud Computing and Data Science*, pp. 15–41, 2024.

[38] T. Laurence, *Blockchain for dummies*. John Wiley & Sons, 2023.

[39] M. Alharby and A. Van Moorsel, “Blockchain-based smart contracts: A systematic mapping study,” *arXiv preprint arXiv:1710.06372*, 2017.

[40] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4.

[41] Y. Freund and R. E. Schapire, "Adaptive boosting," *Machine Learning*, vol. 23, no. 2, pp. 321–357, 1996.

[42] A. Israeli, L. Rokach, and A. Shabtai, "Constraint learning based gradient boosting trees," *Expert Systems with Applications*, vol. 128, pp. 287–300, 2019.

[43] H. Deng, Y. Zhou, L. Wang, and C. Zhang, "Ensemble learning for the early prediction of neonatal jaundice with genetic features," *BMC medical informatics and decision making*, vol. 21, pp. 1–11, 2021.

[44] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[45] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: Unbiased boosting with categorical features," *Advances in neural information processing systems*, vol. 31, 2018.

[46] G. Ke *et al.*, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[47] M. Levi and P. Reuter, "Money laundering," *Crime and justice*, vol. 34, no. 1, pp. 289–375, 2006.

[48] D. Vassallo, V. Vella, and J. Ellul, "Application of gradient boosting algorithms for anti-money laundering in cryptocurrencies," *SN Computer Science*, vol. 2, no. 3, p. 143, 2021.

[49] P. Roszkowska, "Fintech in financial reporting and audit for fraud prevention and safeguarding equity investments," *Journal of Accounting & Organizational Change*, vol. 17, no. 2, pp. 164–196, 2021.

[50] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera, *Learning from imbalanced data sets*. Springer, 2018, vol. 10.

[51] T. Oss and C. E. Budde, "Vulnerability anti-patterns in solidity: Increasing smart contracts security by reducing false alarms," *arXiv preprint arXiv:2410.17204*, 2024, Accessed: 2025-05-02. [Online]. Available: `https://arxiv.org/abs/2410.17204`.

[52] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[53] H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, Ieee, 2008, pp. 1322–1328.

[54] I. Goodfellow *et al.*, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[55] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: A new over-sampling method in imbalanced data sets learning," in *International conference on intelligent computing*, Springer, 2005, pp. 878–887.

[56] S. Srinivasan, A. Vallikannu, L. Manoharan, K. Deepthi, and B. Aravind Yadav, "Identification of the best combination of oversampling technique and machine learning algorithm for credit card fraud detection," in *International Conference on Data Intelligence and Cognitive Informatics*, Springer, 2023, pp. 557–571.

[57] L. Blog, *What is gan? how does it work?* Accessed: 2024-08-25, 2023. [Online]. Available: `https://www.labellerr.com/blog/what-is-gan-how-does-it-work/`.

[58] P. Machado, B. Fernandes, and P. Novais, "Benchmarking data augmentation techniques for tabular data," in *International Conference on Intelligent Data Engineering and Automated Learning*, Springer, 2022, pp. 104–112.

[59] K. G. Al-Hashedi and P. Magalingam, "Financial fraud detection applying data mining techniques: A comprehensive review from 2009 to 2019," *Computer Science Review*, vol. 40, p. 100 402, 2021.

[60] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, vol. 64, no. 3, pp. 107–115, 2021.

[61] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," in *International conference on machine learning*, PMLR, 2016, pp. 1060–1069.

[62] S. Semeniuta, A. Severyn, and S. Gelly, "On accurate evaluation of gans for language generation," *arXiv preprint arXiv:1806.04936*, 2018.

[63] B. Alshawi, "Utilizing gans for credit card fraud detection: A comparison of supervised learning algorithms," *Engineering, Technology & Applied Science Research*, vol. 13, no. 6, pp. 12 264–12 270, 2023.

[64] C. Charitou, A. d. Garcez, and S. Dragicevic, "Semi-supervised gans for fraud detection," in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020, pp. 1–8.

[65] S. Z. Aftabi, A. Ahmadi, and S. Farzi, "Fraud detection in financial statements using data mining and gan models," *Expert Systems with Applications*, vol. 227, p. 120 144, 2023.

[66] F. A. Dael, Ö. Ç. Yavuz, and U. Yavuz, "Stock market prediction using generative adversarial networks (gans): Hybrid intelligent model.," *Comput. Syst. Sci. Eng.*, vol. 47, no. 1, pp. 19–35, 2023.

[67] A. Kozina, "Data transformation review in deep learning," *decision-making*, vol. 7, p. 8, 2024.

[68] M. T. Hannan and N. B. Tuma, "Methods for temporal analysis," *Annual review of sociology*, vol. 5, pp. 303–328, 1979.

[69] M. Carminati, A. Baggio, F. Maggi, U. Spagnolini, and S. Zanero, "Fraudbuster: Temporal analysis and detection of advanced financial frauds," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 15th International Conference, DIMVA 2018, Saclay, France, June 28–29, 2018, Proceedings 15*, Springer, 2018, pp. 211–233.

[70] I. Bose and X. Chen, "Detecting temporal changes in customer behavior," in *2014 International Electrical Engineering Congress (iEECON)*, IEEE, 2014, pp. 1–4.

[71] W. G. Tomek and R. W. Gray, "Temporal relationships among prices on commodity futures markets: Their allocative and stabilizing roles," *American Journal of Agricultural Economics*, vol. 52, no. 3, pp. 372–380, 1970.

[72] I. Sergey, A. Kumar, and A. Hobor, "Temporal properties of smart contracts," in *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice: 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part IV 8*, Springer, 2018, pp. 323–338.

[73] S. Fan, S. Fu, Y. Luo, H. Xu, X. Zhang, and M. Xu, "Smart contract scams detection with topological data analysis on account interaction," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 2022, pp. 468–477.

[74] L. Sevilla-Lara, S. Zha, Z. Yan, V. Goswami, M. Feiszli, and L. Torresani, "Only time can tell: Discovering temporal data for temporal modeling," in *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 2021, pp. 535–544.

[75] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. Cambridge University Press, 2021, `https://d2l.ai/chapter_recurrent-modern/lstm.html`, ISBN: 1009389432. [Online]. Available: `https://d2l.ai/chapter_recurrent-modern/lstm.html` (visited on 10/28/2023).

[76] T. Mikolov, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[77] C. Guo and F. Berkhahn, "Entity embeddings of categorical variables," *arXiv preprint arXiv:1604.06737*, 2016.

[78] J. Li, B. Yang, H. Li, Y. Wang, C. Qi, and Y. Liu, "Dtdr–alstm: Extracting dynamic time-delays to reconstruct multivariate data for improving attention-based lstm industrial time series prediction models," *Knowledge-Based Systems*, vol. 211, p. 106 508, 2021.

[79] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[80] S. Wang, J. Cao, and S. Y. Philip, "Deep learning for spatio-temporal data mining: A survey," *IEEE transactions on knowledge and data engineering*, vol. 34, no. 8, pp. 3681–3700, 2020.

[81] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[82] W. H. L. Pinaya, S. Vieira, R. Garcia-Dias, and A. Mechelli, "Autoencoders," in *Machine learning*, Elsevier, 2020, pp. 193–208.

[83] M. Munir, S. A. Siddiqui, A. Dengel, and S. Ahmed, "Deepant: A deep learning approach for unsupervised anomaly detection in time series," *Ieee Access*, vol. 7, pp. 1991–2005, 2018.

[84] M. Deepa and D. Akila, "Cost-effective anomaly detection for blockchain transactions using unsupervised learning," in *Intelligent Computing and Innovation on Data Science: Proceedings of ICTIDS 2021*, Springer, 2021, pp. 445–453.

[85] A. Bogner, "Seeing is understanding: Anomaly detection in blockchains with visualized features," in *Proceedings of the 2017 ACM international joint conference on pervasive and ubiquitous computing and proceedings of the 2017 ACM international symposium on wearable computers*, 2017, pp. 5–8.

[86] D. Lahat, T. Adali, and C. Jutten, "Multimodal data fusion: An overview of methods, challenges, and prospects," *Proceedings of the IEEE*, vol. 103, no. 9, pp. 1449–1477, 2015.

[87] N. Gaw, S. Yousefi, and M. R. Gahrooei, "Multimodal data fusion for systems improvement: A review," *Handbook of Scholarly Publications from the Air Force Institute of Technology (AFIT), Volume 1, 2000-2020*, pp. 101–136, 2022.

[88] T. Baltrušaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 2, pp. 423–443, 2018.

[89] Y. Zheng, "Methodologies for cross-domain data fusion: An overview," *IEEE transactions on big data*, vol. 1, no. 1, pp. 16–34, 2015.

[90] M. D. Bedworth, "Source diversity and feature-level fusion," in *1999 Information, Decision and Control. Data and Information Fusion Symposium, Signal Processing and Communications Symposium and Decision and Control Symposium. Proceedings (Cat. No. 99EX251)*, IEEE, 1999, pp. 597–602.

[91] K. Liu, Y. Li, N. Xu, and P. Natarajan, "Learn to combine modalities in multimodal deep learning," *arXiv preprint arXiv:1805.11730*, 2018.

[92] E. Androulaki *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, 2018. DOI: 10.1145/3190508.3190538.

[93] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services*, 2016.

[94] Z. Gao, V. Jayasundara, L. Jiang, X. Xia, D. Lo, and J. Grundy, "Smartembed: A tool for clone and bug detection in smart contracts through structural code embedding," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 394–397. DOI: 10.1109/ICSME.2019.00067.

[95] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, IGI global, 2010, pp. 242–264.

[96] T. Looram, L. Nuzzi, and K. Waters, "Reputation oracles: Determining smart contract reputability via transfer learning," *Available at SSRN 4784407*, 2024.

[97] W. Chen, Z. Zheng, J. Cui, E. Ngai, P. Zheng, and Y. Zhou, "Detecting ponzi schemes on ethereum: Towards healthier blockchain technology," in *Proceedings of the 2018 World Wide Web Conference*, Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018, pp. 1409–1418. DOI: 10.1145/3178876.3186046.

[98] Z. Liu, P. Qian, X. Wang, L. Zhu, Q. He, and S. Ji, "Smart contract vulnerability detection: From pure neural network to interpretable graph feature and expert pattern fusion," *arXiv preprint arXiv:2106.09282*, 2021.

[99] X. Tang, Y. Du, A. Lai, Z. Zhang, and L. Shi, "Deep learning-based solution for smart contract vulnerabilities detection," *Scientific Reports*, vol. 13, no. 1, p. 20 106, 2023.

[100] D. Ron and A. Shamir, "Quantitative analysis of the full bitcoin transaction graph," *Financial Cryptography and Data Security*, pp. 6–24, 2013. DOI: 10.1007/978-3-642-39884-1_2.

[101] T. Moore and N. Christin, “Beware the middleman: Empirical analysis of bitcoin-exchange risk,” in *Proceedings of International Conference on Financial Cryptography and Data Security*, Springer, 2013, pp. 25–33.

[102] M. Bartoletti, S. Carta, T. Cimoli, and R. Saia, “Dissecting ponzi schemes on ethereum: Identification, analysis, and impact,” *arXiv preprint arXiv:1703.03779*, 2017.

[103] M. Vasek and T. Moore, “There’s no free lunch, even using bitcoin: Tracking the popularity and profits of virtual currency scams,” in *Proceedings of International Conference on Financial Cryptography and Data Security*, Springer, 2015, pp. 44–61.

[104] K. Christidis and M. Devetsikiotis, “Blockchains and smart contracts for the internet of things,” *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[105] A. Juels, A. Kosba, and E. Shi, “The ring of gyges: Investigating the future of criminal smart contracts,” in *Proceedings of International Conference on Computer and Communications Security*, ACM, 2016, pp. 283–295.

[106] D. Schwartz, N. Youngs, A. Britto, *et al.*, “The ripple protocol consensus algorithm,” *Ripple Labs Inc White Paper*, vol. 5, no. 8, p. 151, 2014.

[107] M. Rossini, *Slither audited smart contracts dataset*, 2022.

[108] R. Agarwal, T. Thapliyal, and S. K. Shukla, “Vulnerability and transaction behavior based detection of malicious smart contracts,” in *Cyberspace Safety and Security: 13th International Symposium, CSS 2021, Virtual Event, November 9–11, 2021, Proceedings 13*, Springer, 2022, pp. 79–96.

[109] L. Duan, L. Yang, C. Liu, W. Ni, and W. Wang, “A new smart contract anomaly detection method by fusing opcode and source code features for blockchain services,” *IEEE Transactions on Network and Service Management*, vol. 20, no. 4, pp. 4354–4368, 2023.

[110] P. L. Rodriguez and A. Spirling, “Word embeddings: What works, what doesn’t, and how to tell the difference for applied research,” *The Journal of Politics*, vol. 84, no. 1, pp. 101–115, 2022.

[111] F. Almeida and G. Xexéo, “Word embeddings: A survey,” *arXiv preprint arXiv:1901.09069*, 2019.

[112] A. Mnih and K. Kavukcuoglu, “Learning word embeddings efficiently with noise-contrastive estimation,” *Advances in neural information processing systems*, vol. 26, 2013.

[113] M. Javaid, A. Haleem, R. P. Singh, R. Suman, and S. Khan, “A review of blockchain technology applications for financial services,” *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, vol. 2, no. 3, p. 100 073, 2022.

[114] X. Ding, Y. Zhang, T. Liu, and J. Duan, "Deep learning for event-driven stock prediction," in *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[115] Martín Abadi *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `https://www.tensorflow.org/`.

[116] Z. Gao, L. Jiang, X. Xia, D. Lo, and J. Grundy, "Checking smart contracts with structural code embedding," *IEEE Transactions on Software Engineering*, vol. 47, no. 12, pp. 2874–2891, 2020.

[117] N. Ashizawa, N. Yanai, J. P. Cruz, and S. Okamura, "Learning contract-wide code representations for vulnerability detection on ethereum smart contracts," *IEICE Technical Report; IEICE Tech. Rep.*, vol. 120, no. 411, pp. 273–280, 2021.

[118] I. Gupta, S. Kumari, P. Jha, and M. Ghosh, "Leveraging lstm and gan for modern malware detection," *arXiv preprint arXiv:2405.04373*, 2024.

[119] SaeStat Teaching, *Section: Variance ratio*, Accessed: 2024-11-30, 2024. [Online]. Available: `https://saestatsteaching.tech/section-varianceratio`.

[120] A. G. Asuero, A. Sayago, and A. González, "The correlation coefficient: An overview," *Critical reviews in analytical chemistry*, vol. 36, no. 1, pp. 41–59, 2006.

[121] Y. Yu, W. Zhang, and Y. Deng, "Frechet inception distance (fid) for evaluating gans," *China University of Mining Technology Beijing Graduate School*, vol. 3, 2021.

[122] K. Kurach, M. Lucic, X. Zhai, M. Michalski, and S. Gelly, "The gan landscape: Losses, architectures, regularization, and normalization," 2018.

[123] A. Radford, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[124] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[125] A. V. Dorogush, V. Ershov, and A. Gulin, "Catboost: Gradient boosting with categorical features support," *arXiv preprint arXiv:1810.11363*, 2018.

[126] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[127] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on knowledge and data engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[128] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[129] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, "Hyperparameter optimization for machine learning models based on bayesian optimization," *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019.

[130] O. Mounnan, O. Manad, L. Boubchir, A. El Mouatasim, and B. Daachi, "A review on deep anomaly detection in blockchain," *Blockchain: Research and Applications*, p. 100 227, 2024.

[131] K. Demertzis, L. Iliadis, N. Tziritas, and P. Kikiras, "Anomaly detection via blockchained deep learning smart contracts in industry 4.0," *Neural Computing and Applications*, vol. 32, no. 23, pp. 17 361–17 378, 2020.

[132] M. U. Hassan, M. H. Rehmani, and J. Chen, "Anomaly detection in blockchain networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 289–318, 2022.

[133] T. Chen *et al.*, "Understanding ethereum via graph analysis," *ACM Transactions on Internet Technology (TOIT)*, vol. 20, no. 2, pp. 1–32, 2020.

[134] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[135] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization.," *Journal of machine learning research*, vol. 13, no. 2, 2012.

[136] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[137] A. Y. Ng, "Feature selection, l 1 vs. l 2 regularization, and rotational invariance," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 78.

[138] A. Borovykh, S. Bohte, and C. W. Oosterlee, "Conditional time series forecasting with convolutional neural networks," *arXiv preprint arXiv:1703.04691*, 2017.

[139] V. Cerqueira, L. Torgo, and I. Mozetič, "Evaluating time series forecasting models: An empirical study on performance estimation methods," *Machine Learning*, vol. 109, no. 11, pp. 1997–2028, 2020.

[140] M. Hasan, M. S. Rahman, H. Janicke, and I. H. Sarker, "Detecting anomalies in blockchain transactions using machine learning classifiers and explainability analysis," *Blockchain: Research and Applications*, p. 100 207, 2024.

[141] Y. Song, Y. Zhu, and F. Wei, "Towards efficient and privacy-preserving anomaly detection of blockchain-based cryptocurrency transactions," in *Information and Communications Security*, D. Wang, M. Yung, Z. Liu, and X. Chen, Eds., Singapore: Springer Nature Singapore, 2023, pp. 590–607, ISBN: 978-981-99-7356-9.

[142] J. Reynolds, L. Sögner, and M. Wagner, "Deviations from triangular arbitrage parity in foreign exchange and bitcoin markets," *Central European Journal of Economic Modelling and Econometrics*, pp. 105–146, 2021.

[143] L. Wei, "Cryptocurrency: Using dark markets to shine light on the propriety of regulation," *U. Ill. JL Tech. & Pol'y*, p. 219, 2022.

[144] F. Zhao, C. Zhang, and B. Geng, "Deep multimodal data fusion," *ACM Computing Surveys*, vol. 56, no. 9, pp. 1–36, 2024.

[145] J. Du, L. Song, T. Zheng, L. Guo, and C. Ma, "A comparative evaluation of autoencoder-based unsupervised anomaly detection methods applied on space payload," in *2019 6th International Conference on Dependable Systems and Their Applications (DSA)*, IEEE, 2020, pp. 269–275.

[146] S. Laridi, G. Palmer, and K.-M. M. Tam, "Enhanced federated anomaly detection through autoencoders using summary statistics-based thresholding," *Scientific Reports*, vol. 14, no. 1, p. 26 704, 2024.

[147] E. A. Wan, "Neural network classification: A bayesian interpretation," *IEEE Transactions on Neural Networks*, vol. 1, no. 4, pp. 303–305, 1990.

[148] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *International workshop on artificial neural networks*, Springer, 1995, pp. 195–201.

[149] A. Ermshaus, P. Schäfer, and U. Leser, "Window size selection in unsupervised time series analytics: A review and benchmark," in *International Workshop on Advanced Analytics and Learning on Temporal Data*, Springer, 2023, pp. 83–101.

[150] D. Hsu, "Multi-period time series modeling with sparsity via bayesian variational inference," *arXiv preprint arXiv:1707.00666*, 2017.

[151] A. Vaswani, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[152] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *International Conference on Learning Representations (ICLR)*, 2015.

Appendix A Complete list of 32 Features

The following table (Table A.1) provides a detailed summary of the transactional data features, formatted to minimize line breaks for shorter descriptions.

Table A.1 Summary of Transactional Data Features used for Smart Contract Analysis.

Column	Description
address	Smart contract address
timeStamp	Timestamp of the transaction
transaction_count_sent	Total number of sent transactions
total_value_sent	Total value sent by the address
avg_value_sent	Average value sent per transaction
min_value_sent	Minimum value sent
max_value_sent	Maximum value sent
gas_sum_sent	Total gas used for sent transactions
gas_mean_sent	Average gas used for sent transactions
gasPrice_sum_sent	Total gas price for sent transactions
gasUsed_sum_sent	Total gas used for sent transactions
gasUsed_mean_sent	Average gas used per sent transaction
error_count_sent	Number of errors in sent transactions

(continued...)

Column	Description
unique_contracts_interacted_with_sent	Unique contracts interacted with (sent)
avg_min_between_sent_tnx	Average minutes between sent transactions
transaction_count_received	Total number of received transactions
total_value_received	Total value received by the address
avg_value_received	Average value received per transaction
min_value_received	Minimum value received
max_value_received	Maximum value received
gas_sum_received	Total gas used for received transactions
gas_mean_received	Average gas used for received transactions
gasPrice_sum_received	Total gas price for received transactions
gasUsed_sum_received	Total gas used for received transactions
gasUsed_mean_received	Average gas used per received transaction
error_count_received	Number of errors in received transactions
unique_contracts_interacted_with_received	Unique contracts interacted with (received)
avg_min_between_received_tnx	Average minutes between received transactions
time_diff_between_first_and_last	Time difference between the first and last transaction

(continued...)

Column	Description
avg_min_between_sent_tnx_missing	Average minutes between sent transactions (missing)
avg_min_between_received_tnx_missing	Average minutes between received transactions (missing)
rep_score	Reputability Score

Appendix B Python Libraries Version

Python Library	Version
Python	3.10.16
catboost	1.2.5
cudf	24.08.02
cupy	13.3.0
imblearn	0.12.3
lightgbm	4.5.0
matplotlib	3.9.2
numpy	1.26.4
pandas	2.2.2
pyevmasm	0.2.3
ruptures	1.1.9
scikit-learn	1.5.1
seaborn	0.13.2
scipy	1.14.1
tensorflow	2.18.0
xgboost	1.7.5

Table B.1 Python Libraries Version

Appendix C Supplementary Results of Boosted Embedd based Bytecode Analysis

C.1 Opcode Categories

Category	Opcodes
ARITH	ADD, MUL, SUB, DIV, SDIV, SMOD, MOD, ADDMOD, MULMOD, EXP, SIGNEXTEND
COMPARE	LT, GT, SLT, SGT
LOGIC	AND, OR, XOR, NOT
MEMORY	MLOAD, MSTORE, SLOAD, SSTORE, MSIZE
RETURN	RETURN, REVERT, RETURNDATASIZE, RETURNDATACOPY
PUSH	PUSH1, PUSH2, PUSH3, ..., PUSH32
DUP	DUP1, DUP2, DUP3, ..., DUP16
SWAP	SWAP1, SWAP2, SWAP3, ..., SWAP16
LOG	LOG0, LOG1, LOG2, LOG3, LOG4

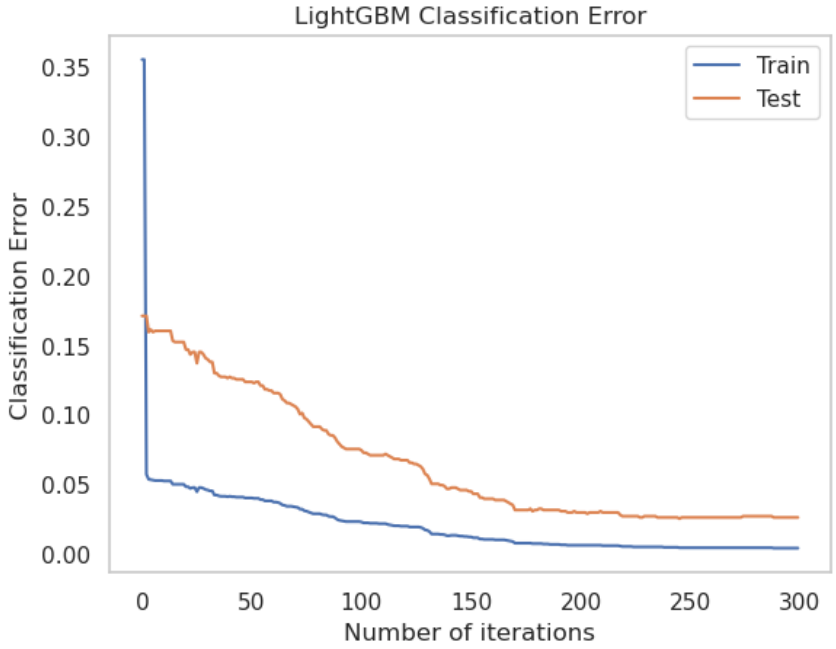
Table C.1 Opcode categories and their associated instructions.

C.2 Classification Report

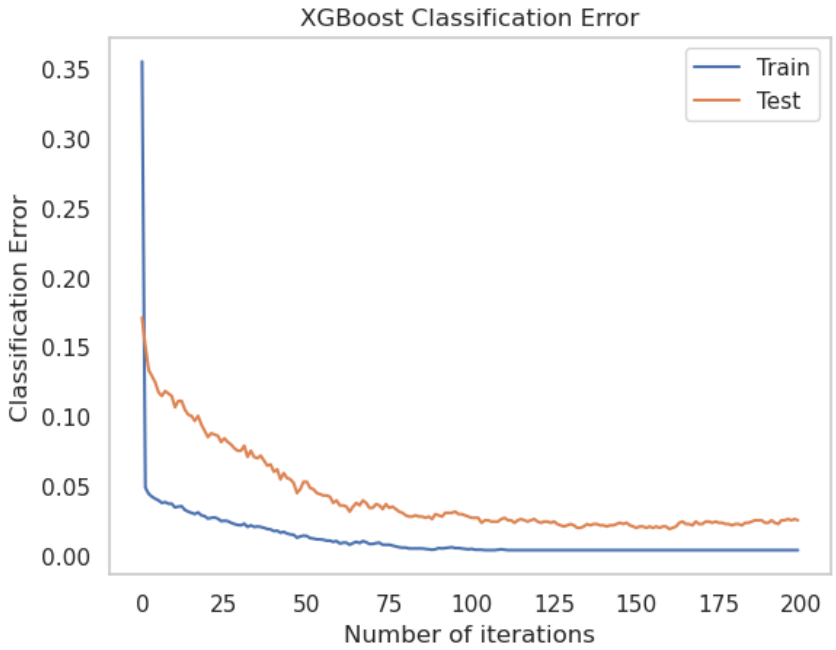
Algorithm	Class	Precision	Recall	F1-Score
XGBoost	Illicit	0.9137	0.9424	0.9278
	Reputable	0.9880	0.9816	0.9848
LightGBM	Illicit	0.9231	0.9424	0.9326
	Reputable	0.9881	0.9838	0.9859
AdaBoost	Illicit	0.8621	0.9162	0.8883
	Reputable	0.9825	0.9698	0.9761

Table C.2 Detailed Classification Metrics for Boosting Algorithms

C.3 Classification Errors of LGBM and XGBoost



(a) Classification Error against No. of Iterations of LGBM



(b) Classification Error against No. of Iterations of XGBoost

Figure C.1 Classification Error against No. of Iterations for LGBM and XGBoost across 5-Fold Cross Validation.

Appendix D Autoencoder Hyperparmeters and Supplementary Results

Hyperparameter	Description	Value (Batch 16)	Value (Batch 32)	Value (Batch 64)
Batch Size	Number of samples processed before the model is updated.	16	32	64
LSTM Units 1	Number of units in the first LSTM layer.	64	32	16
LSTM Units 2	Number of units in the second LSTM layer.	16	64	8
Learning Rate	Step size for each iteration to reach the loss minimum.	0.000310	0.000229	0.006859
Activation Function	Function that introduces non-linearity to the model.	ReLU	ReLU	Tanh
L2 Regularization 1	Penalty term to reduce overfitting.	0.08	0.04	0.01
L2 Regularization 2	Penalty term for the second layer.	0.06	0.09	0.06
Dropout Rate 1	Fraction of inputs dropped during training.	0.3	0.0	0.4
Dropout Rate 2	Fraction of inputs dropped in the second layer.	0.2	0.4	0.1
Validation Loss	Loss on the validation dataset during training.	0.018201	0.018858	0.021692

Table D.1 Complete list of Optimal Hyperparameters for Transactions-Only LSTM-Based Autoencoder

Hyperparameter	Description	Value (Batch 16)	Value (Batch 32)	Value (Batch 64)
Batch Size	Number of samples processed before the model is updated.	16	32	64
Best Filters 1	Number of filters in the first convolutional layer.	64	64	16
Best Filters 2	Number of filters in the second convolutional layer.	8	24	32
Learning Rate	Step size for each iteration to reach the loss minimum.	0.002968	0.001953	0.000759
Activation Function	Function that introduces non-linearity to the model.	Tanh	Tanh	ReLU
Kernel Size 1	Dimensions of the filter used in the first layer.	5	3	5
Kernel Size 2	Dimensions of the filter used in the second layer.	5	5	3
L2 Regularization 1	Penalty term to reduce overfitting.	0.0	0.01	0.0
L2 Regularization 2	Penalty term for the second layer.	0.0	0.08	0.03
Pool Size	Size of the pooling layer for down-sampling.	4	4	2
Dropout Rate 1	Fraction of inputs dropped during training.	0.2	0.4	0.2
Dropout Rate 2	Fraction of inputs dropped in the second layer.	0.3	0.4	0.3
Up Sampling Size 1	Size of the upsampling layer for feature expansion.	2	4	2
Up Sampling Size 2	Size of the upsampling layer for further expansion.	4	4	4
Validation Loss	Loss on the validation dataset during training.	0.010588	0.021718	0.016061

Table D.2 Complete list of Optimal Hyperparameter for Transactions-Only CNN-Based Autoencoder

Hyperparameter	Description	Value (Batch 16)	Value (Batch 32)	Value (Batch 64)
Batch Size	Number of samples processed before the model is updated.	16	32	64
Best Filters 1	Number of filters in the first convolutional layer.	48	48	16
Best Filters 2	Number of filters in the second convolutional layer.	24	16	8
Learning Rate	Step size for each iteration to reach the loss minimum.	0.001207	0.001632	0.002350
Activation Function	Function that introduces non-linearity to the model.	ReLU	Tanh	ReLU
Kernel Size 1	Dimensions of the filter used in the first layer.	3	3	5
Kernel Size 2	Dimensions of the filter used in the second layer.	3	5	5
L2 Regularization 1	Penalty term to reduce overfitting.	0.0	0.03	0.01
L2 Regularization 2	Penalty term for the second layer.	0.1	0.03	0.01
Pool Size 1	Size of the pooling layer for down-sampling.	2	2	2
Pool Size 2	Size of the second pooling layer.	2	4	4
Dropout Rate 1	Fraction of inputs dropped during training.	0.2	0.3	0.0
Dropout Rate 2	Fraction of inputs dropped in the second layer.	0.1	0.4	0.2
Up Sampling Size 1	Size of the upsampling layer for feature expansion.	4	4	4
Up Sampling Size 2	Size of the upsampling layer for further expansion.	4	4	4
Validation Loss	Loss on the validation dataset during training.	0.006271	0.008100	0.006751

Table D.3 Complete list of Optimal Hyperparameters for Multimodal CAE

Appendix E Detailed Model Configuration and Validation Results for Experiment 3

Table E.1 Model Configurations for LSTM: Average MAE, Best Validation Loss, and Best Epoch Across Folds

LSTM Units	Batch Size	Learning Rate	MAE	Best Val Loss	Best Epoch
32	32	0.001	0.1611	0.04335	16
32	32	0.0005	0.1636	0.04399	21
32	32	0.0001	0.1617	0.04338	20
32	64	0.001	0.1651	0.04386	19
32	64	0.0005	0.1602	0.04346	22
32	64	0.0001	0.1608	0.04339	21
64	32	0.001	0.1591	0.04306	27
64	32	0.0005	0.1595	0.04325	24
64	32	0.0001	0.1593	0.04310	27
64	64	0.001	0.1597	0.04334	20
64	64	0.0005	0.1666	0.04397	16
64	64	0.0001	0.1599	0.04328	25
128	32	0.001	0.1608	0.04363	25
128	32	0.0005	0.1588	0.04298	28
128	32	0.0001	0.1594	0.04315	28
128	64	0.001	0.1599	0.04311	27
128	64	0.0005	0.1587	0.04317	25
128	64	0.0001	0.1595	0.04323	29

Table E.2 Multihead Attention-Based LSTM-CNN Configruations and Val Loss by Fold and Trial: Validation Loss, Filter and LSTM Units, Learning Rate, Activation, Dropout Rate, and L2 Regularization

Fold	Trial	Validation Loss	Num Filters	LSTM Units	Learning Rate	Activation	Dropout Rate	L2 Regularization
1	07	0.04486	64	32	0.0005	tanh	0.1	0.0001
1	01	0.04524	128	64	0.0001	tanh	0.1	0.0001
1	06	0.04539	128	32	0.001	tanh	0.3	0.001
1	09	0.04542	32	32	0.001	tanh	0.1	0.0001
1	05	0.04558	128	32	0.0005	tanh	0.3	0.001
1	02	0.04572	32	32	0.0005	relu	0.2	0.001
1	08	0.04574	64	32	0.0005	tanh	0.1	0.01
1	00	0.04617	64	128	0.0005	relu	0.3	0.001
1	04	0.04720	128	128	0.0001	relu	0.3	0.001
1	03	0.04792	64	128	0.001	relu	0.2	0.01
2	07	0.04486	64	32	0.0005	tanh	0.1	0.0001
2	01	0.04524	128	64	0.0001	tanh	0.1	0.0001
2	06	0.04539	128	32	0.001	tanh	0.3	0.001
2	09	0.04542	32	32	0.001	tanh	0.1	0.0001
2	05	0.04558	128	32	0.0005	tanh	0.3	0.001
2	02	0.04572	32	32	0.0005	relu	0.2	0.001
2	08	0.04574	64	32	0.0005	tanh	0.1	0.01
2	00	0.04617	64	128	0.0005	relu	0.3	0.001
2	04	0.04720	128	128	0.0001	relu	0.3	0.001

continued on next page

Fold	Trial	Validation Loss	Num Filters	LSTM Units	Learning Rate	Activation	Dropout Rate	L2 Regularization
2	03	0.04792	64	128	0.001	relu	0.2	0.01
3	07	0.04486	64	32	0.0005	tanh	0.1	0.0001
3	01	0.04524	128	64	0.0001	tanh	0.1	0.0001
3	06	0.04539	128	32	0.001	tanh	0.3	0.001
3	09	0.04542	32	32	0.001	tanh	0.1	0.0001
3	05	0.04558	128	32	0.0005	tanh	0.3	0.001
3	02	0.04572	32	32	0.0005	relu	0.2	0.001
3	08	0.04574	64	32	0.0005	tanh	0.1	0.01
3	00	0.04617	64	128	0.0005	relu	0.3	0.001
3	04	0.04720	128	128	0.0001	relu	0.3	0.001
3	03	0.04792	64	128	0.001	relu	0.2	0.01