https://doi.org/10.35662/unine-thesis-3240

Université de Neuchâtel
Institut d'informatique

Practical Machine Learning for Decentralized Finance and Smart Contract Security

Ph.D. dissertation

submitted to the Faculty of Science of the University of Neuchâtel

to attain the degree of Doctor of Philosophy in Computer Science

by

Pasquale De Rosa

Approved by the dissertation committee :

Prof. tit. Valerio Schiavoni	thesis co-director and reviewer • University of Neuchâtel, Switzerland
Prof. Pascal Felber	thesis co-director • University of Neuchâtel, Switzerland
Prof. Miguel Matos	reviewer • University of Lisbon, Portugal
Prof. Giuseppe Di Luna	reviewer • University of Rome La Sapienza, Italy
Prof. Lydia Chen	expert • University of Neuchâtel, Switzerland

Defended on December 11, 2025

Université de Neuchâtel
Faculté des sciences

Rue Emile-Argand 11
CH-2000 Neuchâtel
doctorat.sciences@unine.ch

IMPRIMATUR POUR THESE DE DOCTORAT

La Faculté des sciences de l'Université de Neuchâtel autorise l'impression de la présente thèse soutenue par

Monsieur Pasquale DE ROSA

Titre :

"Practical Machine Learning for Decentralized Finance and Smart Contract Security"

sur le rapport des membres du jury composé comme suit :

- **Prof. tit. Valerio Schiavoni,** co-directeur de thèse, Université de Neuchâtel, Suisse
- **Prof. Pascal Felber,** co-directeur de thèse, Université de Neuchâtel, Suisse
- **Prof. Miguel Matos,** Universidade de Lisboa, Portugal
- **Prof. Giuseppe Di Luna,** Sapienza Università di Roma, Italie
- **Prof. Lydia Chen,** Université de Neuchâtel, Suisse

Neuchâtel, le 9 février 2026

Le Doyen, Prof. P. Brunner

To my parents, Sabato and Patrizia.
To my sister, Antonia.
To my grandma, Antonietta.
To my late grandpa, Pasquale.
And to our little dog, Grogu.
Thank you for your love, for believing in me, and for always being there for me.

Abstract

This dissertation examines the design, implementation, and evaluation of reproducible Machine Learning (ML) systems across two distinct yet conceptually connected domains: Decentralized Finance (DeFi) and Smart Contract (SC) security. In both areas, the research aims to translate theoretical insights into operational frameworks that are transparent, data-driven, and adaptable to real-world deployment.

The first part focuses on the dynamics of cryptocoin markets, characterized by high volatility and strong interdependencies among assets. Building on empirical evidence of persistent cross-correlations, the study develops and validates correlation-driven forecasting models that exploit related coin trajectories as predictive signals. These insights culminate in the design of CryptoAnalytics, an open-source Python toolkit that automates the entire forecasting pipeline, from data collection and preprocessing to model training, validation, and deployment. By integrating Gradient-Boosting Machines (GBMs) and Recurrent Neural Networks (RNNs) within a modular architecture, CryptoAnalytics bridges research reproducibility with operational applicability, supporting both academic experimentation and production-grade prediction services.

The second part turns to blockchain security and the detection of malicious SCs in decentralized ecosystems. Here, the dissertation introduces PhishingHook, the first reproducible framework for detecting phishing contracts on Ethereum through static opcode-level analysis. PhishingHook unifies dataset construction, bytecode disassembly, and model benchmarking within a single environment, evaluating sixteen ML and Deep Learning (DL) models across multiple families. Results demonstrate that static opcode representations achieve high accuracy (around 90%) without relying on user data or runtime traces, while remaining robust to temporal performance decay. Comprehensive statistical analysis and post hoc interpretability further ensure transparency and reliability in the evaluation.

In addition, this dissertation presents the first systematic study of Ethereum Virtual Machine (EVM) bytecode obfuscation and its impact on SC malware detection. Through an extensive evaluation of two obfuscation tools across 27 transformation configurations, the analysis reveals how structural rewriting affects bytecode validity, semantic preservation, gas cost, and the robustness of diverse detection models. The results highlight significant limitations in current obfuscation techniques and demonstrate that vision-based classifiers remain notably resilient under aggressive control-flow distortion.

Overall, this dissertation demonstrates how reproducible ML systems can bridge the gap between theoretical modeling and operational deployment. In the financial domain, they enable interpretable and data-driven forecasting; in blockchain security, they establish practical foundations for proactive and transparent threat detection. By unifying these contributions under a shared methodological philosophy, the work advances the broader goal of developing open, reliable, and adaptive intelligent systems for data-intensive, decentralized environments.

Keywords: Machine Learning, Decentralized Finance, Smart Contracts, Model Serving, Financial Forecasting, Malware Detection.

Résumé

Cette thèse étudie la conception, l'implémentation et l'évaluation de systèmes d'Apprentissage Automatique (ML) reproductibles dans deux domaines distincts mais conceptuellement liés : la finance décentralisée (DeFi) et la sécurité des contrats intelligents (SCs). Dans ces deux contextes, la recherche vise à traduire des fondements théoriques en cadres opérationnels transparents, fondés sur les données et adaptés aux environnements réels.

La première partie analyse la dynamique des marchés de cryptomonnaies, marqués par une forte volatilité et de fortes interdépendances entre actifs. À partir de preuves empiriques de corrélations croisées persistantes, l'étude développe et valide des modèles de prévision exploitant les trajectoires de monnaies liées comme variables prédictives. Ces travaux conduisent à la conception de CryptoAnalytics, une boîte à outils Python open source automatisant l'ensemble de la chaîne de prévision, de la collecte et du prétraitement des données au déploiement des modèles. En intégrant des GBMs et des RNNs dans une architecture modulaire, CryptoAnalytics relie reproductibilité de la recherche et applicabilité opérationnelle, soutenant l'expérimentation académique comme les services de prévision en production.

La deuxième partie porte sur la sécurité des blockchains et la détection de contrats intelligents malveillants. Elle introduit PhishingHook, premier cadre reproductible de détection de contrats d'hameçonnage sur Ethereum par analyse statique au niveau opcode. PhishingHook unifie la construction de jeux de données, le désassemblage du bytecode et l'évaluation de seize modèles de ML et de DL de différentes familles au sein d'un environnement unique. Les résultats montrent que les représentations statiques des opcodes atteignent une précision élevée (environ 90%) sans dépendre des données utilisateurs ni des traces d'exécution, tout en restant robustes face à la dégradation temporelle. Une analyse statistique approfondie et des méthodes d'interprétabilité post hoc garantissent en outre la transparence et la fiabilité des évaluations.

En complément, cette thèse présente la première étude systématique de l'obfuscation du bytecode EVM et de son impact sur la détection de logiciels malveillants dans les contrats intelligents. À partir d'une évaluation de deux outils d'obfuscation couvrant 27 configurations, l'analyse montre comment la réécriture structurelle affecte la validité du bytecode, la préservation sémantique, le coût en gaz et la robustesse des modèles de détection. Les résultats révèlent les limites des techniques actuelles et indiquent que les classifieurs fondés sur des représentations visuelles restent particulièrement résilients face à des distorsions agressives du flot de contrôle.

Cette thèse montre comment des systèmes de ML reproductibles relient modélisation théorique et déploiement opérationnel, en permettant une prévision interprétable en finance et une détection proactive et transparente des menaces en sécurité des blockchains. En unifiant ces contributions sous une philosophie méthodologique commune, elle contribue au développement de systèmes intelligents ouverts, fiables et adaptatifs pour des environnements décentralisés et riches en données.

Mots-clés : Apprentissage Automatique, Finance Décentralisée, Contrats Intelligents, Déploiement de Modèles, Prévision Financière, Détection de Logiciels Malveillants.

Acknowledgements

This thesis represents the culmination of four intense and rewarding years of work, growth, and dedication. Yet it would not exist without the extraordinary people and places that supported me throughout this journey. I therefore begin by expressing my deepest gratitude to Switzerland, and to Neuchâtel in particular, for welcoming me with openness, kindness, and opportunities far beyond what I had imagined. I arrived in this country as a 25-year-old; I leave six years later, two spent at the University of Geneva and four at the University of Neuchâtel, as a more mature, slightly grey-haired 31-year-old researcher, profoundly grateful and enriched both professionally and personally.

My heartfelt thanks go first to my supervisors and mentors, Valerio Schiavoni and Pascal Felber. They taught me not only how to conduct research, but also how to think critically, ask meaningful questions, and share knowledge responsibly.

Valerio guided me from my very first day, when I was still extremely green. Together we challenged ourselves immediately by writing a paper during my first month, work that was later accepted at DAIS in Lucca in 2022. His energy, passion, and relentless curiosity have been a constant source of inspiration, and I owe a large part of my growth as a researcher to his extraordinary guidance. Simply put, none of this would have happened without him.

Pascal has been an exceptional mentor and a thoughtful guide in my teaching journey, particularly in the programming and compiler courses. I vividly remember starting my first classes feeling intimidated by teaching in French and convinced it would be a major challenge. I could not have been more wrong. Over time, I came to genuinely love teaching, and I owe this transformation largely to his trust, encouragement, and example. From him, I learned what it truly means to be an educator: to listen, to support students in their difficulties, and to care deeply about their success.

Together, Valerio and Pascal instilled in me a vision of research that goes beyond publications: pairing every paper with an open-source artifact, and striving to make knowledge accessible, reproducible, and useful. This philosophy of openness and responsibility is among the most valuable lessons I will carry with me.

I am also deeply grateful to the exceptional collaborators and co-authors I had the pleasure to work with outside the University of Neuchâtel: Sara Bouchenak and her wonderful team at INSA Lyon, where I carried out a research stay while working on our IMWUT 2023 paper; Yérom-David Bromberg (University of Rennes) for his guidance and immense help on the IC2E 2024 and DSN 2025 papers; and the outstanding cybersecurity team at the University of Rome ”La Sapienza”: Leonardo Querzoni, Giuseppe Antonio Di Luna, and Daniele Cono D’Elia. I also warmly thank Miguel Matos for kindly accepting to serve on the jury of my Ph.D. defense.

My gratitude extends to all the colleagues who made my time in Switzerland both productive and joyful. At the University of Geneva, I thank Nils Schaetti and Marjolaine Yvorra for sup-

porting my first transition from industry to academia, and my dear friend Simon Queyrut, who later followed me to Neuchâtel as a fellow Ph.D. student.

At the University of Neuchâtel, I had the privilege of working with an incredible group of people. From the ”old guard”: Rémi Dulong, Sébastien Vaucher, Christian Goettel, Catherine Ikae, Jämes Ménétrey, and Peterson Yuhala. From the ”new guard”: Abele Malan, Andreas Athanasopoulos, Gert Lek, Basile Lewandowski, Mpoki Mwaisela, Jakub ”Kuba” Tluczek, Victor Villin, Louis Vialar, Elif Yilmaz, Hortence Yiepnou, Giulio Segalini, and my great office mate, Romain De Laage. You created an environment that was stimulating, supportive, and genuinely fun, and I am grateful for every discussion, collaboration, and shared coffee.

I also wish to thank the new professors, Lydia Chen and Christos Dimitrakakis, as well as the senior professors, now retired. In particular, Jacques Savoy, my first mentor in teaching, who guided me through my very first classes in statistics with R and digital humanities; and Peter Kropf, with whom I spent countless wonderful hours talking, learning, and receiving thoughtful advice at exactly the right moments. A special thank-you as well to Marcelo Pasin, for the many laughs, great conversations, and that unforgettable road trip to Lucca for DAIS 2022.

I am grateful to the wider university community, especially the IMI team, Vladimir Macko and Abdessalam Ouaazki, for our enjoyable collaboration on a programming-class project, as well as to all visiting researchers, particularly my fellow Italians and dear friends Andrea De Murtas, Lorenzo Brescia, and Salvatore Busiello.

Beyond academia, I thank the friends who made Switzerland feel like home: Loris Mombelli, Lisa Seccia, Cristina Landolina, Iva Stratoberdha, and Giulia Gaggero. I am also grateful to my lifelong friends Angelo Foggia, Stefano Vergara, Fernando D’Antò, and Davide Iazzetta, whose presence and friendship have endured despite distance. A special thanks goes to my childhood friend, long-time travel companion, and personal trainer, Luca Perrotta, for countless adventures and constant motivation.

Finally, and most importantly, I thank my family. Even when my choices took me far from home, I have always felt your unconditional support and unwavering belief in me.

My dad, Sabato, for supporting me in every possible way and always respecting my path. My mum, Patrizia, for her endless patience, strength, and love. My sister, Antonia, now a 26-year-old young woman, but forever my little sister, who means more to me than words can express. My grandma, Antonietta, for her warmth and the tenderness she has always given me. My grandpa, Pasquale, who left us fifteen years ago and could not witness this Ph.D., but whose values continue to guide me every day.

And finally, even if he cannot read (being a toy poodle), the latest arrival: our sweet angel, Grogu.

Contents

List of Acronyms

AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
NLP	Natural Language Processing
HSC	Histogram Similarity Classifier
CVM	Computer Vision Model
LLM	Large Language Model
VDM	Vulnerability Detection Model
MLaaS	Machine Learning as a Service
MLOps	Machine Learning Operations
API	Application Programming Interface
CPU	Central Processing Unit
GPU	Graphics Processing Unit
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
OOB	Out-of-Bag
SGD	Stochastic Gradient Descent
ReLU	Rectified Linear Unit
tanh	Hyperbolic Tangent
ONNX	Open Neural Network Exchange
AUT	Area Under Time
FNN	Feedforward Neural Network
MLP	Multi-Layer Perceptron
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
GBM	Gradient-Boosting Machine
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
MA	Moving Average
ARIMA	Autoregressive Integrated Moving Average
VAR	Vector Autoregression
OLS	Ordinary Least Squares
MLE	Maximum Likelihood Estimation
RSS	Residual Sum of Squares
GLM	Generalized Linear Model
LR	Linear Regression
MLR	Multiple Linear Regression
LogReg	Logistic Regression
KNN	K-Nearest Neighbors
SVM	Support Vector Machine
CART	Classification and Regression Tree
RF	Random Forest
XGBoost	Extreme Gradient Boosting
CatBoost	Categorical Boosting
LightGBM	Light Gradient Boosting Machine
GOSS	Gradient-based One-Side Sampling
EFB	Exclusive Feature Bundling
ViT	Vision Transformer
GPT-2	Generative Pretrained Transformer 2
T5	Text-to-Text Transfer Transformer
ECA	Efficient Channel Attention
CAM	Class Activation Mapping
GNN	Graph Neural Network
GCN	Graph Convolutional Network
GAT	Graph Attention Network
GIN	Graph Isomorphism Network
TAG	Topology Adaptive Graph Convolutional Network
GraphSAGE	Graph Sample and Aggregate
DeFi	Decentralized Finance
SC	Smart Contract
dApp	Decentralized Application
EVM	Ethereum Virtual Machine
WASM	WebAssembly
DAO	Decentralized Autonomous Organization
CFG	Control Flow Graph
OHLC	Open-High-Low-Close
SPV	Simplified Payment Verification
MPT	Merkle Patricia Trie
PoW	Proof-of-Work

PoS	Proof-of-Stake
FFG	Friendly Finality Gadget
OHLC	Open-High-Low-Close
BTC	Bitcoin
ETH	Ether
LTC	Litecoin
ADA	Cardano
XRP	Ripple
BNB	Binance Coin
USDT	Tether
USDC	USD Coin
DOGE	Dogecoin
SOL	Solana
XLM	Stellar Lumens
DASH	Dash
XMR	Monero
BCH	Bitcoin Cash
BAT	Basic Attention Token
LINK	Chainlink
NEO	Neo
QTUM	Defiance Quantum ETF
TRX	Tronix
ZEC	Zcash
BSV	Bitcoin Satoshi Vision
BORA	BORA Token
EOS	EOS
BUSD	Binance USD
T-Y	Toda-Yamamoto
m-Wald	modified Wald
KPSS	Kwiatkowski-Phillips-Schmidt-Shin
ADF	Augmented Dickey-Fuller
AIC	Akaike Information Criterion
S-W	Shapiro-Wilk
K-W	Kruskal-Wallis
CDD	Critical Difference Diagram
GED	Graph Edit Distance
APK	Android Package
REST	Representational State Transfer
gRPC	Google Remote Procedure Call
MAR	Model Archive
JPEG	Joint Photographic Experts Group
CLI	Command-Line Interface

List of Figures

List of Tables

Chapter 1.

Introduction

This chapter outlines the motivation and scope of the thesis, focusing on the role of Machine Learning in Decentralized Finance and Smart Contract security. It highlights the main contributions on model-serving strategies, cryptocoin price forecasting, and detection of malicious contracts, while also outlining the overall structure of the dissertation along with the related publications and open-source software artifacts.

Chapter Outline

1.1. Context and Motivation

Artificial Intelligence (AI), and in particular Machine Learning (ML), have become central to modern computing. Unlike traditional programming with fixed rules, ML systems learn patterns from data to make predictions or decisions [1]. The search for patterns in data has long been a cornerstone of scientific progress. For example, astronomical observations in the 16th century enabled Kepler to derive the laws of planetary motion [2], and the identification of regularities in atomic spectra in the early 20th century was instrumental in the development of quantum physics [3–5]. Building on this tradition, ML provides computational methods for uncovering structure in data and leveraging it for tasks such as classification, forecasting, and decision-making. This paradigm is now deeply embedded in contemporary technologies: streaming platforms recommending movies [6, 7], navigation systems predicting traffic conditions [8, 9], smartphones recognizing faces in photos [10, 11], and financial institutions assessing risks and market dynamics [12, 13].

While these examples illustrate the maturity of ML in consumer services, new domains are emerging where the stakes are considerably higher. Financial technologies and blockchain-based systems, in particular, increasingly rely on ML for automation, risk assessment, and security [14, 15]. In these settings, errors or vulnerabilities can result directly in substantial monetary losses or systemic risks, making the practical deployment of ML both more challenging and more critical. Prominent examples include Decentralized Finance (DeFi) platforms [16], which provide global-scale financial services built on top of cryptocoins such as Bitcoin (BTC) [17] and Ether (ETH) [18], and Smart Contracts (SCs) [19], which execute agreements automatically without trusted intermediaries. Both domains highlight the need for practical and trustworthy ML solutions.

The rapid growth of DeFi has led to a global market capitalization comparable to national economies [20], largely driven by assets such as BTC and ETH. However, trading prices remain extremely volatile, with sudden and often unpredictable fluctuations [21, 22]. Understanding co-movement [23] and cross-correlation [24] patterns among crypto-assets is therefore essential for constructing reliable forecasting models that can support investors, institutions, and regulators in navigating this dynamic ecosystem.

At the same time, the programmability of blockchains through SCs exposes billions of dollars in assets to malicious activity [25–28]. Phishing, scams, and other forms of SC-based malware cause substantial financial damage every year [29, 30]. Adversaries increasingly employ code obfuscation [31–34], undermining the effectiveness of existing detection techniques. This underscores the need for robust, reproducible ML-based frameworks capable of proactively detecting malicious SCs, even under adversarial conditions.

Finally, practical adoption of these solutions depends on efficient deployment of ML models in real-world environments. While training is inherently resource-intensive [35, 36], inference must be executed under strict latency and cost constraints [37]. In financial applications, forecasts lose relevance if delivered after market shifts, whereas in security, delayed detection can enable irreversible blockchain transactions. Commercial Machine Learning as

a Service (MLaaS) platforms [38] offered by major cloud providers deliver powerful deployment capabilities [39, 40], but remain costly [41–43] and raise concerns regarding control and trust [44, 45]. Open-source model-serving frameworks [46–50] and optimized inference strategies thus play a critical enabling role, ensuring that ML methods for DeFi and SC security can be deployed at scale.

By addressing challenges of performance, forecasting, and protection, this thesis contributes to making ML both practical and trustworthy in blockchain-based financial systems.

1.2. Thesis Contributions

This thesis advances the practical use of ML in DeFi and SC security, with a focus on efficiency, robustness, and reproducibility. The contributions span three complementary directions: *(i)* strategies for efficient ML inference, *(ii)* forecasting and analysis of cryptocoin markets in DeFi, and *(iii)* proactive detection of malicious SCs. The main contributions are summarized below.

Benchmarking ML Model-Serving Frameworks

We present a systematic evaluation of model-serving frameworks [51], comparing five state-of-the-art open-source solutions (TensorFlow Serving [46], TorchServe [47], MLServer [50], MLflow [49], BentoML [48]) across four representative workloads. Our study focuses on inference performance at the serving layer and highlights trade-offs between Deep Learning (DL)-specific servers and general-purpose frameworks, providing actionable insights for deployment. This contribution establishes the basis for later parts of the thesis by addressing efficiency and reproducibility in real-world inference.

Forecasting Cryptocoin Prices from Correlation Patterns

We analyze the dynamics of the cryptocoin market [52, 53] by studying correlation and causality among major assets such as BTC, ETH, and a large set of altcoins. Building on these observations, we develop CryptoAnalytics [54], a modular forecasting framework that integrates Recurrent Neural Networks (RNNs) and Gradient-Boosting Machines (GBMs) to capture cross-asset dependencies. The framework achieves competitive accuracy in predicting price trends, substantially outperforming simple mean and median regression baselines across all error metrics (MSE, RMSE, MAE, and MAPE), and is extended into a deployable service, narrowing the gap between experimental research and practical financial applications.

Detecting Malicious Smart Contracts through Bytecode Analysis

We design PhishingHook [55, 56], the first evaluation framework for phishing detection in Ethereum SCs using opcode analysis. The framework supports dataset construction, model comparison, and post-hoc analysis and is benchmarked with 16 distinct detection techniques, including Histogram Similarity Classifiers (HSCs), Computer Vision Models (CVMs), and Large

Language Models (LLMs). We further construct and release the largest labeled dataset of phishing contracts on Ethereum. In complementary work [57], we conduct the first systematic study of Ethereum Virtual Machine (EVM) bytecode obfuscation and its impact on malware detection, exposing critical weaknesses in existing models and showing the relative resilience of vision-based approaches.

Releasing Datasets, Models, and Tools for Reproducibility

All code, datasets, trained models, and experimental artifacts are released through public repositories (GitHub and Zenodo) [58–62]. This commitment to open science ensures full reproducibility of results and provides the community with concrete tools to advance research in ML for finance and blockchain security.

1.3. Thesis Outline

This thesis is organised into four parts, presenting the research contributions in a logical order for clarity rather than strict chronology.

Part I introduces the foundations of this work. Chapter 2 reviews the principles of ML with a focus on model architectures, while Chapter 3 presents the concepts of blockchains, DeFi, and SC that motivate the subsequent contributions.

Part II explores strategies for efficient ML inference. Chapter 4 provides a systematic benchmarking of five state-of-the-art model-serving frameworks across representative workloads, highlighting trade-offs between DL-specific and general-purpose solutions.

Part III discusses the application of ML techniques to DeFi. Chapters 5 and 6 analyse correlation and causality patterns among major cryptocoins and introduce CryptoAnalytics, a modular forecasting framework that achieves competitive accuracy in predicting price trends.

Part IV addresses SC security. Chapter 7 introduces PhishingHook, the first evaluation framework for phishing detection in Ethereum SCs and presents the largest labeled dataset of phishing contracts to date. Chapter 8 evaluates the impact of bytecode obfuscation on malware detection, exposing limitations of current approaches and highlighting the robustness of vision-based techniques.

Finally, Chapter 9 concludes this dissertation by summarising the key contributions and discussing future research directions in the application of ML to DeFi and SC security.

1.4. Publications

The core contributions of this thesis are presented in the following papers:

- **Pasquale De Rosa**, Pascal Felber and Valerio Schiavoni. *Seeing Through EVM Bytecode Obfuscation*. In IEEE Access, 2026. https://doi.org/10.1109/ACCESS.2026.3662238.

- **Pasquale De Rosa**, Simon Queyrut, Yérom-David Bromberg, Pascal Felber and Valerio Schiavoni. *PhishingHook: Catching Phishing Ethereum Smart Contracts leveraging EVM Opcodes*. In 55th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Naples, Italy, June 23-26, 2025. https://doi.org/10.1109/DSN64029.2025.00033.

- **Pasquale De Rosa**, Simon Queyrut, Yérom-David Bromberg, Pascal Felber and Valerio Schiavoni. *PhishingHook: Catching Phishing Ethereum Smart Contracts leveraging EVM Opcodes*. In 55th IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S), Naples, Italy, June 23-26, 2025. https://doi.org/10.1109/DSN-S65789.2025.00075.

- **Pasquale De Rosa**, Pascal Felber and Valerio Schiavoni. *ScamDetect: Towards a Robust, Agnostic Framework to Uncover Threats in Smart Contracts*. In 55th IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S), Naples, Italy, June 23-26, 2025. https://doi.org/10.1109/DSN-S65789.2025.00068.

- **Pasquale De Rosa**, Pascal Felber and Valerio Schiavoni. *CryptoAnalytics: Cryptocoins price forecasting with machine learning techniques*. In SoftwareX, vol. 26, 101663, 2024. https://doi.org/10.1016/j.softx.2024.101663.

- **Pasquale De Rosa**, Yérom-David Bromberg, Pascal Felber, Djob Mvondo and Valerio Schiavoni. *On the Cost of Model-Serving Frameworks: An Experimental Evaluation*. In 12th IEEE International Conference on Cloud Engineering (IC2E), Paphos, Cyprus, September 24-27, 2024. https://doi.org/10.1109/IC2E61754.2024.00032.

- **Pasquale De Rosa**, Pascal Felber and Valerio Schiavoni. *Practical Forecasting of Cryptocoins Time-series using Correlation Patterns*. In 17th ACM International Conference on Distributed and Event-based Systems (DEBS), Neuchâtel, Switzerland, June 27-30, 2023. https://doi.org/10.1145/3583678.3596888.

- **Pasquale De Rosa** and Valerio Schiavoni. *Understanding Cryptocoins Trends Correlations*. In 22nd IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS), Lucca, Italy, June 13-17, 2022. https://doi.org/10.1007/978-3-031-16092-9_3.

In addition, the following paper was published during my doctoral studies. Although it is not part of the core contributions of this thesis, it is related to the broader scope of my research:

- Yasmine Djebrouni, Nawel Benarba, Ousmane Touat, **Pasquale De Rosa**, Sara Bouchenak, Angela Bonifati, Pascal Felber, Vania Marangozova and Valerio Schiavoni. *Bias Mitigation in Federated Learning for Edge Computing*. In Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT), vol. 7, no. 4, 2023. https://doi.org/10.1145/3631455.

1.5. Open-Source Software Contributions

As part of this doctoral research, several open-source software artifacts were developed to support the experiments and methodologies presented in my studies. To ensure long-term accessibility and reproducibility, these artifacts have been made publicly available on GitHub or archived on Zenodo. The corresponding repositories are listed below:

- **Pasquale De Rosa**, Pascal Felber and Valerio Schiavoni. *Seeing Through EVM Bytecode Obfuscation*. Zenodo, 2026. https://doi.org/10.5281/zenodo.15660760.

- **Pasquale De Rosa**, Simon Queyrut, Yérom-David Bromberg, Pascal Felber and Valerio Schiavoni. *PhishingHook: Catching Phishing Ethereum Smart Contracts leveraging EVM Opcodes*. Zenodo, 2025. https://doi.org/10.5281/zenodo.14933708.

- **Pasquale De Rosa**, Yérom-David Bromberg, Pascal Felber, Djob Mvondo and Valerio Schiavoni. *On the Cost of Model-Serving Frameworks: An Experimental Evaluation*. Zenodo, 2024. https://doi.org/10.5281/zenodo.11200983.

- **Pasquale De Rosa**, Pascal Felber and Valerio Schiavoni. *CryptoAnalytics: Cryptocoins Price Forecasting with Machine Learning Techniques*. GitHub, 2024. https://github.com/quapsale/cryptoanalytics-software.

- **Pasquale De Rosa**, Pascal Felber and Valerio Schiavoni. *Cryptocoins Analytics: Analysis and Forecasting of cryptocoin Price Trends*. GitHub, 2023. https://github.com/quapsale/cryptoanalytics.

- Yasmine Djebrouni, Nawel Benarba, Ousmane Touat, **Pasquale De Rosa**, Sara Bouchenak, Angela Bonifati, Pascal Felber, Vania Marangozova and Valerio Schiavoni. *ASTRAL: Accurate Bias Mitigation at Scale in Federated Learning*. GitHub, 2023. https://github.com/FL-Bias/ASTRAL.

Part I.

Background

Chapter 2.

Foundations of Machine Learning

This chapter introduces the fundamental concepts and techniques of Machine Learning that underpin the methods used throughout this thesis. It reviews traditional approaches such as linear models, instance-based methods, Support Vector Machines, and tree-based ensembles, before presenting modern Deep Learning architectures including Feedforward Neural Networks, Convolutional Neural Networks, Recurrent Neural Networks, and Transformer models. The goal is to provide the theoretical background necessary to understand the models and algorithms applied in subsequent chapters.

Chapter Outline

2.1. Traditional Machine Learning

Machine Learning (ML) [1, 63, 64] refers to a class of algorithms that improve their performance at a given task through exposure to data. A widely cited definition by Mitchell [65] states: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E". This formulation emphasizes that learning is not an end in itself, but rather a means of acquiring the ability to perform a task more effectively. For instance, if the goal is to enable a robot to walk, the task is walking; learning provides a systematic way of reaching that capability, as opposed to explicitly programming every movement.

From a practical perspective, ML is compelling because it allows us to address problems that are too complex to solve with hand-crafted rules. At a more conceptual level, it is also linked to our understanding of intelligence, since many principles underlying ML are closely tied to how intelligent behavior can emerge [63].

In most ML settings, data is represented as a set of examples, each described by a vector of features. Formally, an input example can be written as $x \in \mathbb{R}^n$, where each component x_i corresponds to a measurable attribute of the underlying object or event. In images, for example, the features are typically the pixel intensity values [63].

A wide range of tasks can be formulated within this framework. Two of the most common ones are:

- **Classification**: Here the objective is to assign an input to one of k discrete categories. The learning algorithm estimates a function $f : \mathbb{R}^n \to \{1, \ldots, k\}$, mapping feature vectors to class labels. In some variants, f produces a probability distribution over classes rather than a single label. A canonical example is object recognition, where an image is classified according to the object it depicts. Such methods have applications ranging from robotic perception to face recognition in consumer software [63].
- **Regression**: In regression tasks, the goal is to predict a continuous output variable. The learner models a function $f : \mathbb{R}^n \to \mathbb{R}$ that maps inputs to numerical values. Examples include forecasting financial time series or estimating the expected insurance claims of a policyholder. These predictions are central to domains like risk assessment and algorithmic trading [63].

These fundamental task types underpin the field of ML. A variety of algorithmic paradigms have been developed to address them, including linear and instance-based methods as well as tree-based and ensemble approaches. The following subsections review these classical techniques in more detail.

2.1.1. Linear and Instance-Based Methods

Among the earliest and most widely used tools in ML are linear models. Their enduring appeal lies in their simplicity, interpretability, and often surprisingly strong performance across a wide range of tasks. Despite the rise of highly flexible approaches such as neural networks, Linear

Regression (LR) [66] and its close relatives remain foundational techniques, frequently serving both as baselines and as interpretable components within more complex systems [64].

Linear Regression

LR models the relationship between a predictor X and an outcome Y as approximately linear [64]:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

Here, β_0 is the intercept, β_1 the slope, and ε an error term capturing variation not explained by the linear trend. For example, if X denotes expenditure on television advertising and Y denotes sales, then β_1 represents the average increase in sales per additional unit of advertising [64].

The coefficients are estimated by choosing the line that minimizes the total squared differences between observed and predicted values, the Residual Sum of Squares (RSS) [64, 67]. This fitted line can then be used both for prediction and to interpret the strength and direction of the association between X and Y.

Figure 2.1 illustrates simple LR of Y on X. The red line is the Ordinary Least Squares (OLS) fit $\hat{y} = 1.96 + 0.52x$, obtained by minimizing the RSS. The scatter of points around the line reflects residual variation not captured by the linear trend.

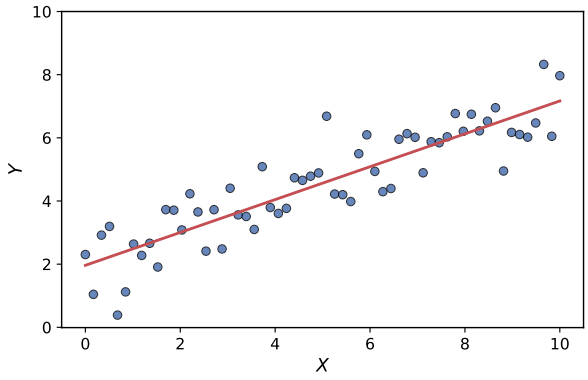

Figure 2.1.: Linear Regression of Y on X, showing observed data points (blue) and the fitted regression line (red). Estimated coefficients are $\hat{\beta}_0 = 1.96$ and $\hat{\beta}_1 = 0.52$.

In practice, outcomes are rarely determined by a single factor. Multiple Linear Regression (MLR) generalizes this framework to include multiple predictors [64]:

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon$$

Here, each coefficient β_j represents the expected change in Y from a one-unit increase in X_j, while holding all other predictors constant.

LR remains widely used for its interpretability and efficiency. Estimated coefficients provide simple summaries of predictor-response relationships, although the method relies on assumptions, such as linearity and additivity, that are rarely met exactly in empirical data [64]. Moreover,

when predictors are highly correlated (multicollinearity), coefficient estimates can become unstable, with inflated standard errors and possibly incorrect signs [68].

Despite these limitations, LR serves as a cornerstone for more advanced methodologies. It illustrates core ideas in estimation, inference, and model evaluation that recur throughout ML, and forms the basis for Generalized Linear Models (GLMs) [69]. Among these, Logistic Regression (LogReg) [70] is particularly prominent, replacing the continuous outcome with a probabilistic framework suited to classification tasks. This extension will be the focus of the following section.

Logistic Regression

While LR provides a natural starting point for modeling relationships between predictors and outcomes, it is not well suited when the response is categorical. In particular, applying a linear model to binary outcomes can yield nonsensical predictions outside the valid probability range $[0, 1]$. LogReg [70] addresses this by directly modeling the probability of class membership through a nonlinear transformation [64].

Formally, let the response be binary, $Y \in \{0, 1\}$, with predictors $X \in \mathbb{R}^p$. The goal is to model the conditional probability of $Y = 1$:

$$p(x) = \Pr(Y = 1 \mid x)$$

Instead of assuming a linear relationship between $p(X)$ and X, LogReg employs the logistic function [71]:

$$\hat{p}(x) = \frac{e^{\hat{\beta}_0 + \sum_{j=1}^{p} \hat{\beta}_j x_j}}{1 + e^{\hat{\beta}_0 + \sum_{j=1}^{p} \hat{\beta}_j x_j}}$$

which maps any linear combination of predictors to the valid probability range $[0, 1]$. A common decision rule classifies an observation as class 1 when $\hat{p}(x) > 0.5$ [64].

This model can also be viewed in terms of log-odds. The odds of $Y = 1$ are $\hat{p}(x)/(1 - \hat{p}(x))$, and taking logarithms gives the logit form:

$$\log\left(\frac{\hat{p}(x)}{1 - \hat{p}(x)}\right) = \hat{\beta}_0 + \sum_{j=1}^{p} \hat{\beta}_j x_j$$

Thus, LogReg is a linear model on the log-odds scale: a one-unit increase in X_j changes the log-odds of $Y = 1$ by β_j, or equivalently multiplies the odds by e^{β_j} [64].

Model parameters are estimated via Maximum Likelihood Estimation (MLE) [72], which chooses coefficients that maximize the likelihood of the observed data. Unlike LR, this optimization has no closed-form solution, but efficient iterative algorithms are well established [73].

Figure 2.2 illustrates LogReg of Y on X. The red curve represents the fitted logistic function $\hat{p}(x)$, with coefficients estimated by MLE. Blue points denote observed binary outcomes plotted against their predictor values, and the dashed line at $p = 0.5$ marks the decision threshold separating the two classes.

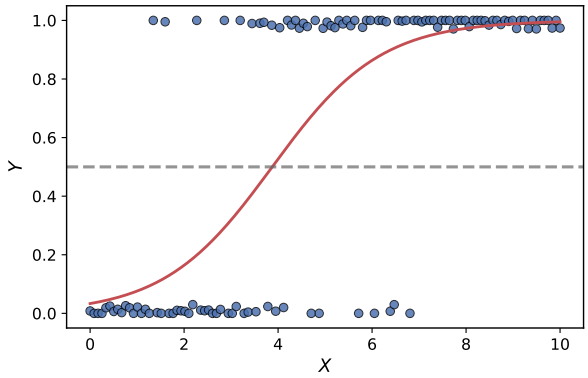

Figure 2.2.: Logistic Regression of Y on X. Observed data points are shown in blue, with the fitted logistic curve in red. Estimated coefficients are $\hat{\beta}_0 = -3.37$ and $\hat{\beta}_1 = 0.87$.

Despite its simplicity, LogReg remains one of the most widely applied methods in practice. It extends linear modeling to classification tasks by introducing a probabilistic framework with a nonlinear link between predictors and outcomes. Like LR, it is valued for its interpretability and efficiency, but it also inherits the limitations of parametric models, relying on assumptions such as independence of errors, linearity of continuous predictors in the logit, absence of multicollinearity, and the lack of strongly influential outliers [74].

An alternative is to avoid explicit parametric assumptions and instead base predictions directly on the training data. Such instance-based methods defer generalization until a query is made, relying on similarity measures between input examples. Among these, the K-Nearest Neighbors (KNN) algorithm [75] is one of the most intuitive and widely used [64].

K-Nearest Neighbors

The KNN algorithm [75] is a simple and intuitive example of instance-based learning. Given a query point x_0, the method identifies the K training points closest to x_0, denoted by $\mathcal{N}(x_0)$.

In classification, the conditional probability of class j is estimated as the fraction of neighbors belonging to that class [64]:

$$\hat{p}_j(x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}(x_0)} \mathbf{1}(y_i = j)$$

where $\mathbf{1}(y_i = j) = 1$ if neighbor i belongs to class j and 0 otherwise. The predicted class is then chosen as the majority class among the K neighbors.

The performance of the classifier depends critically on the choice of K. With $K = 1$, decision boundaries are highly irregular, interpolating the training data with zero training error but very high variance. As K increases, the boundaries become smoother and less variable, reducing variance at the cost of increased bias. Intermediate K values often yield the best test performance by balancing underfitting and overfitting [64]. Figure 2.3 illustrates this effect for $K = 1$ and $K = 100$.

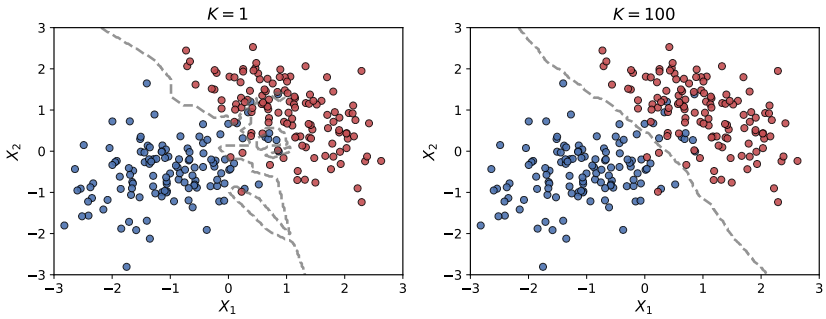

Figure 2.3.: Decision boundaries from K-Nearest Neighbors classification with $K = 1$ and $K = 100$. Data points are in blue ($Y = 0$) and red ($Y = 1$), with boundaries in gray.

The same principle extends naturally to regression. For a query point x_0, the algorithm predicts the response as the average of the K nearest neighbors [64]:

$$\hat{f}(x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}(x_0)} y_i$$

When $K = 1$, the regression fit interpolates the training data exactly, producing a step function with high variance. As K increases, the fit becomes smoother and more biased, since predictions average over broader neighborhoods. Figure 2.4 shows this effect for $K = 1$ and $K = 100$.

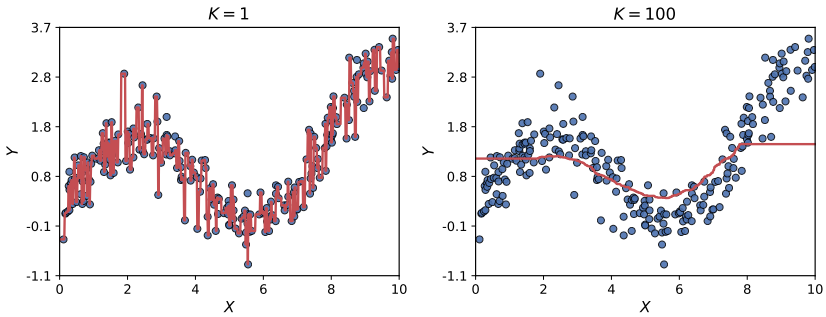

Figure 2.4.: K-Nearest Neighbors regression with $K = 1$ (left) and $K = 100$ (right). Observed data points are shown in blue, with fitted regression curves in red.

Both classification and regression with KNN highlight the fundamental bias-variance tradeoff [76]: small K values provide flexible fits with low bias but high variance, while large K values reduce variance through averaging but increase bias by oversmoothing the data. Although computationally intensive at prediction time and sensitive to the choice of distance metric, KNN remains a widely used and instructive example of nonparametric, instance-based learning [64].

2.1.2. Support Vector Machines

While linear models and instance-based methods provide important tools for classification, they can struggle when the decision boundary is highly non-linear. Support Vector Machines (SVMs) [77] offer a principled framework to address this challenge.

At their core, SVMs are large-margin classifiers: they seek a separating boundary between classes that maximizes the margin, the distance to the nearest training points. A larger margin makes the classifier more robust to noise and promotes better generalization by preferring simpler, lower-capacity decision rules [64]. The training points lying exactly on the margin are called support vectors, and they alone determine the final classifier. In the linearly separable case, these support vectors uniquely define the separating hyperplane [64].

In many real-world problems, class boundaries are not straight. One way to capture non-linear boundaries is to expand the feature space with polynomial or interaction terms so that a linear boundary in the expanded space corresponds to a curved boundary in the original space. Explicitly constructing such features, however, is often computationally expensive [64].

SVMs resolve this difficulty through the kernel trick. The decision function depends only on inner products between observations, which can be replaced by a kernel function $K(\cdot,\cdot)$:

$$\hat{f}(x) = \hat{\beta}_0 + \sum_{i \in S} \hat{\alpha}_i K(x, x_i)$$

where S indexes the support vectors and $\hat{\alpha}_i$ are learned coefficients. Using kernels allows the method to behave as if operating in a higher-dimensional feature space, without explicitly computing those new coordinates [64].

Several kernels are commonly used in practice:

- Linear kernel [77], the standard inner product $K(x, x') = x^\top x'$, producing a linear decision boundary.
- Polynomial kernel [77], $K(x, x') = (1 + x^\top x')^d$, which introduces curved boundaries, with flexibility controlled by the degree d.
- Radial basis function (RBF) kernel [78], emphasizes local similarity:

  $$K(x, x') = \mathrm{e}^{-\gamma \sum_{j=1}^{p} (x_j - x'_j)^2}$$

  where $\gamma > 0$ determines how quickly similarity decays with distance.

Figure 2.5 illustrates decision boundaries obtained with different kernels. Both the polynomial and RBF kernels can capture complex patterns that a linear separator would miss.

As with other methods, flexibility involves a bias-variance trade-off. Highly flexible kernels (for example, an RBF with large γ) can overfit by tailoring boundaries too closely to the training data, while overly rigid kernels may underfit. In addition, SVMs introduce a regularization parameter $C > 0$ that controls tolerance for margin violations. Large C penalizes misclassifications heavily, producing narrower margins that fit the training data closely, while small C allows wider

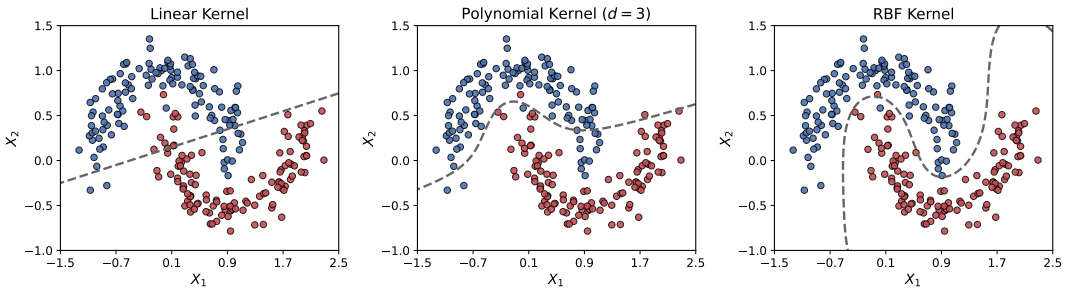

Figure 2.5.: Support Vector Machines with different kernels. Observed data points are shown in blue ($Y = 0$) and red ($Y = 1$), with dashed gray decision boundaries.

margins with more training errors. In this way, C balances fitting the data against maintaining a robust margin [77].

Despite the need for careful tuning of C and kernel parameters, SVMs remain widely used in practice. They are valued for their strong theoretical foundations, robustness to noise, and ability to model complex, non-linear relationships between predictors and outcomes [64].

2.1.3. Tree-Based and Ensemble Methods

While linear models, instance-based approaches, and margin-based classifiers such as SVMs provide powerful tools, they can be limited when the relationship between predictors and outcomes is highly complex or involves intricate interactions among variables. Tree-based methods offer an alternative perspective: instead of fitting coefficients or computing similarities, they recursively partition the feature space into regions and make predictions based on the training observations that fall into each region [64]. The resulting models, commonly referred to as decision trees [79], are appealing for their interpretability and their ability to naturally capture nonlinear relationships and high-order interactions without explicit feature engineering.

On their own, individual trees often suffer from high variance and may not achieve state-of-the-art predictive accuracy. However, when used as building blocks within ensemble methods, they form the foundation of some of the most powerful modern algorithms [64]. Techniques such as Random Forests (RFs) [80] combine many trees in structured ways to substantially improve stability and accuracy, albeit at the cost of reduced interpretability.

The following subsections review these approaches in turn. We begin with Classification and Regression Trees (CARTs) [81], then describe ensemble methods that leverage trees as base learners, focusing on bagging [82] and RFs, and finally turn to boosting [83].

Classification and Regression Trees

CARTs [81] are among the most intuitive modeling approaches in ML. Their core idea is to partition the predictor space into a set of simple, non-overlapping regions and then make predictions based on the training observations within each region [64]. Unlike linear models, which assume additive effects, trees approximate the response surface by stratification: they

divide the feature space into axis-aligned regions, and predictions within each terminal region (leaf) are constant [64].

Formally, a regression tree partitions the input space X into J disjoint regions $R_1, \ldots, R_J$. For any test point $x \in R_j$, the prediction is the mean of the training responses in that region:

$$\hat{f}(x) = \frac{1}{|R_j|} \sum_{i:x_i \in R_j} y_i$$

These regions are constructed via recursive binary splitting, a greedy procedure that at each step selects the predictor and threshold producing the largest reduction in prediction error [64]. This process continues within each resulting region until a stopping criterion is met (for example, a minimum number of observations per leaf).

Figure 2.6 shows an example regression tree trained on the Diabetes dataset [84]. Internal nodes display the selected biomedical predictor and its threshold, while the leaves report the predicted progression score.

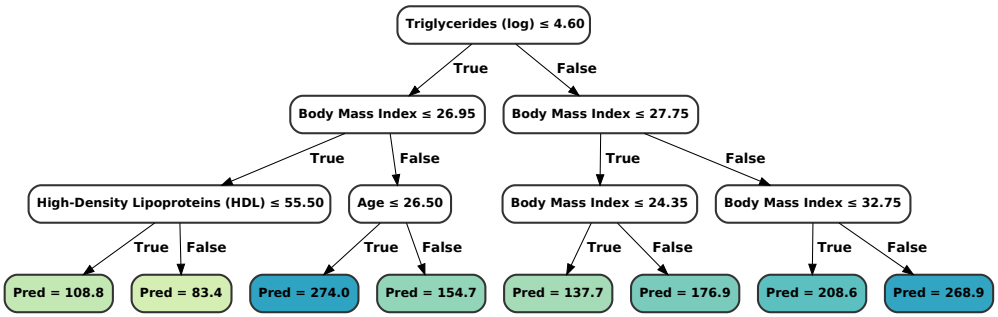

Figure 2.6.: Regression tree trained on the Diabetes dataset [84]. Internal nodes show biomedical predictors and leaves give estimated disease progression scores.

Classification trees apply the same principle to categorical outcomes. Instead of minimizing squared error, they choose splits that maximize node purity, aiming to make each region as homogeneous as possible with respect to class labels [64]. A commonly used measure of node impurity is the Gini index [85]:

$$G(R_m) = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

where $\hat{p}_{mk}$ is the proportion of training observations in node m that belong to class k. Another popular criterion is the cross-entropy [86], $D(R_m) = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$. Both measures are minimized when a node is pure, *i.e.*, when all observations in the node belong to the same class.

Figure 2.7 shows an example classification tree trained on the Iris dataset [87]. Internal nodes show which measurement and threshold are used to split the data, and each leaf assigns a species label.

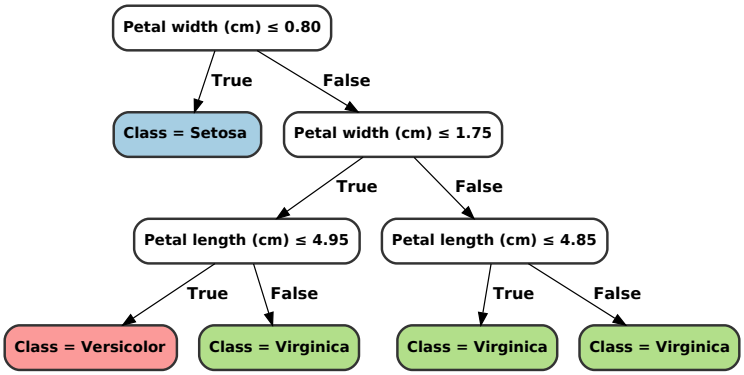

Figure 2.7.: Classification tree trained on the Iris dataset [87]. Internal nodes show petal measurements (cm), and leaves assign species labels (Setosa, Versicolor, Virginica).

Trees offer several advantages over linear models: they are highly interpretable, can naturally handle both quantitative and qualitative predictors, and capture nonlinear interactions without explicit feature construction. However, single trees are typically less accurate than other methods due to their high variance [64]. This motivates ensemble approaches, where many trees are aggregated to produce stronger and more stable predictors. The next section introduces these methods, beginning with bagging and RFs, and continuing with boosting.

Ensemble Methods: Bagging and Boosting

Although CARTs offer flexibility and interpretability, their predictive performance is often hindered by high variance: even small perturbations in the training data can produce markedly different tree structures and predictions [64]. Ensemble methods mitigate this instability by combining many trees into a single, more robust predictor. Among these, two of the most widely used are bagging [82] and boosting [83], which differ fundamentally in how the individual trees are constructed and how their outputs are aggregated.

Bagging, short for bootstrap aggregation, reduces variance by averaging over trees trained on different bootstrap samples of the original dataset [82]. Concretely, B bootstrap samples are drawn from the training data, and a decision tree is fit to each sample. For a new input x_0, the bagged prediction in regression is:

$$\hat{f}_{\text{bag}}(x_0) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b^*(x_0)$$

while in classification the predicted class is chosen by majority vote. This averaging operation substantially reduces variance without inflating bias. Bagging also provides a natural estimate

of test error through the Out-of-Bag (OOB) observations, which are left out of each bootstrap sample and can be used to assess predictive accuracy [82].

While bagging reduces variance, the resulting trees can still be highly correlated if certain strong predictors dominate the splits. RFs [80] address this by randomly selecting only a subset of predictors at each split, which decorrelates the trees and further improves ensemble performance. Like bagging, they aggregate predictions by averaging for regression and majority vote for classification, and they remain widely used for their robustness, minimal tuning requirements, and built-in measures of variable importance [64].

Boosting [83] takes a different approach. Instead of fitting many trees in parallel and averaging them, boosting builds trees sequentially, with each new tree focusing on the errors made by the current ensemble. Small trees (often stumps) are fit to the residual errors, and their predictions are added to the model using a small learning rate λ:

$$\hat{f}_{\text{boost}}(x) = \sum_{b=1}^{B} \lambda \hat{f}_b(x)$$

This stagewise procedure gradually shifts the model's attention to harder-to-predict observations, reducing bias while controlling variance through λ and the number of boosting iterations B [64].

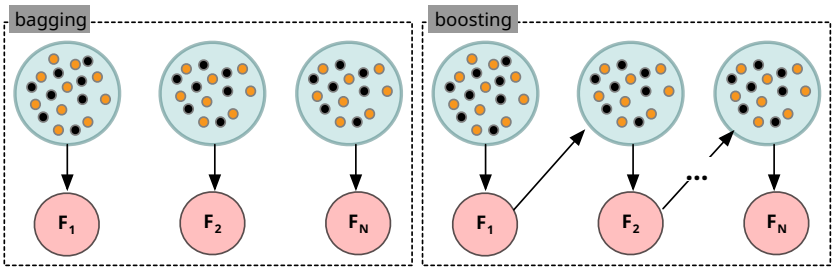

Figure 2.8.: Comparison of ensemble methods: bagging builds learners in parallel on resampled data, while boosting trains them sequentially, focusing on errors [53].

Gradient boosting [88] extends traditional boosting by determining each new weak learner through gradient-based optimization of a chosen loss function. At each stage, a tree is fit to the negative gradient of the loss with respect to the current model's predictions, gradually reducing the overall error. Because of this formulation, models built using this approach are known as Gradient-Boosting Machines (GBMs), or Gradient-Boosted Trees (GBTs). They have become one of the most widely used and competitive model families for structured data.

Several state-of-the-art implementations of GBMs are widely used in practice and will be employed later in this work:

- Extreme Gradient Boosting (XGBoost) [89] is an open-source, scalable, and distributed GBM framework. It introduced regularized objective functions, parallel tree construction, and efficient handling of sparse data, making it suitable for a wide range of machine learning tasks. XGBoost has powered numerous winning solutions in data science competitions

(*e.g.*, 17 of 29 winning entries in Kaggle's KDDCup 2015) and has also been successfully applied beyond traditional ML domains, such as energy physics and particle research [90].

- Light Gradient Boosting Machine (LightGBM) [91], developed by Microsoft, improves training efficiency using two key techniques. First, Gradient-based One-Side Sampling (GOSS) speeds up training on large datasets by keeping the samples with the largest gradient values (the hardest to predict) and randomly sampling from the rest, preserving accuracy while reducing data per iteration. Second, Exclusive Feature Bundling (EFB) groups sparse, mutually exclusive features into single combined features to reduce dimensionality. These techniques give LightGBM fast training speed, low memory usage, and strong scalability on large datasets.
- Categorical Boosting (CatBoost) [92], developed by Yandex, addresses a common problem in gradient boosting known as prediction shift, where models inadvertently use target values from the same data they are predicting, leading to target leakage and overly optimistic results. CatBoost avoids this by using ordered boosting, which builds trees using only earlier (past) observations when computing target statistics. It also handles categorical variables natively, reducing the need for extensive preprocessing and often improving accuracy on datasets with many categorical features.

These modern GBM frameworks share the same underlying principle, sequentially combining many shallow trees to form a strong ensemble, but differ in how they optimize computation, incorporate regularization, and handle large or heterogeneous datasets. Their scalability, efficiency, and consistently strong predictive performance have made them a standard choice for many real-world ML applications [53–56].

2.2. Deep Learning Architectures

While the traditional ML methods introduced in Section 2.1 have proven effective across a wide range of tasks, their performance can degrade when confronted with high-dimensional, unstructured data such as images, audio, or natural language. Such data often exhibit rich hierarchical structure that is difficult to capture using hand-crafted features or shallow models. Deep Learning (DL) [63] addresses this challenge by learning multiple levels of representation directly from raw data through deep neural architectures.

At a high level, DL models consist of layers of simple computational units (neurons) connected by weighted edges. Each layer transforms its input representation into a more abstract one: early layers detect low-level patterns, while deeper layers compose them into increasingly complex concepts. This hierarchical feature learning enables DL to excel at tasks involving perception and sequential reasoning [63].

The following subsections review the main classes of deep architectures used in this thesis: Feedforward Neural Networks (FNNs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformer models.

2.2.1. Feedforward Neural Networks

One of the earliest neural models is the perceptron [93], which maps an input vector to a binary output through a single layer of weighted connections and a step activation function. While historically important, a single perceptron can only represent linearly separable boundaries.

FNNs, also called Multi-Layer Perceptrons (MLPs), generalize this idea by stacking multiple layers of nonlinear transformations between input and output [63]. Figure 2.9 contrasts a single-layer perceptron with an MLP: while the former can only represent linear functions, the latter can approximate arbitrarily complex nonlinear mappings, as guaranteed by the universal approximation theorem [94].

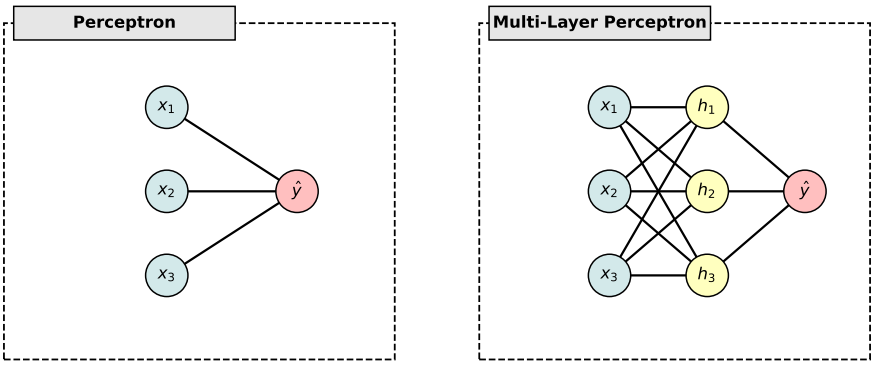

Figure 2.9.: Single-layer perceptron vs. multi-layer perceptron. Stacking multiple layers enables learning of complex nonlinear functions.

Formally, an MLP with L layers computes successive hidden representations

$$h^{(0)} = x, \qquad h^{(\ell)} = \sigma\big(W^{(\ell)} h^{(\ell-1)} + b^{(\ell)}\big) \quad (\ell = 1, \dots, L)$$

where $x \in \mathbb{R}^n$ is the input, $W^{(\ell)}$ and $b^{(\ell)}$ are learnable weights and biases, and $\sigma(\cdot)$ is a nonlinear activation such as the Rectified Linear Unit (ReLU) [95] or the Hyperbolic Tangent (tanh) [96]). The output is $f(x) = h^{(L)}$.

Training proceeds by minimizing a task-specific loss (for example, cross-entropy for classification or Mean Squared Error (MSE) for regression) using Stochastic Gradient Descent (SGD) [97]. Backpropagation [98] efficiently computes gradients via the chain rule[1].

Although conceptually simple, MLPs provide the building blocks for more specialized architectures. Their dense connectivity allows them to model complex functions, but it also makes them parameter-intensive and prone to overfitting on high-dimensional inputs such as images. This motivates architectures that incorporate structural priors, such as spatial locality or temporal order.

[1]The chain rule states that the derivative of a composition of functions is the product of the derivatives of the composed functions.

2.2.2. Convolutional Neural Networks

CNNs [96, 99] are designed to process data with grid-like topology, such as images. Instead of connecting every neuron to every input as in an MLP, CNNs use small local filters that slide across the input and compute weighted combinations of neighboring values. The same filter is applied at all spatial locations, enabling parameter sharing and translation invariance while drastically reducing the number of learnable parameters.

A typical CNN stacks several convolutional layers, often followed by pooling layers that downsample the feature maps, reducing spatial resolution and making the representation more compact. After this feature extraction stage, the resulting maps are flattened and passed through fully connected layers to produce the final predictions [63]. Figure 2.10 illustrates this structure.

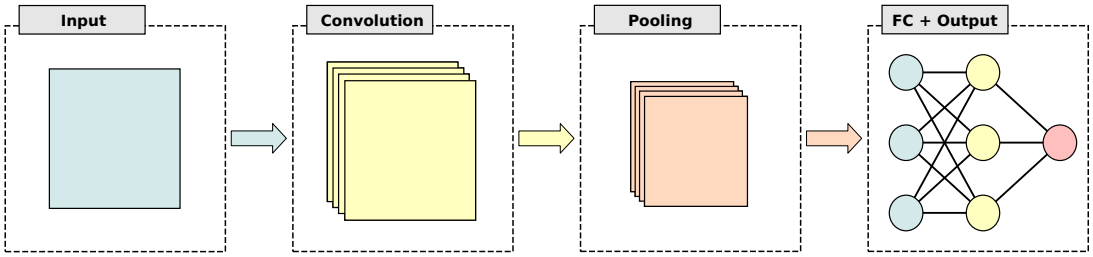

Figure 2.10.: Simplified CNN architecture. Convolutional and pooling layers extract spatial features, which are fed into fully connected layers for classification.

Convolutional layers can be written schematically as

$$h = \sigma(W * x + b)$$

where $*$ denotes the discrete convolution, x is the input feature map, W the filter kernel, and σ an activation function (typically ReLU).

Modern CNNs often combine efficient backbone architectures with lightweight attention modules. For example, EfficientNet [100] introduced a principled compound scaling rule that jointly adjusts network depth, width, and resolution to maximize accuracy for a given computational budget, while the ECA module [101] adaptively reweights channels using a fast 1D convolution without dimensionality reduction. These advances illustrate how CNNs have evolved to balance accuracy, efficiency, and generalization.

This architectural bias for spatial locality has made CNNs the dominant approach in Computer Vision tasks. However, they are less suited to sequential data, where temporal dependencies are crucial. Recurrent architectures address this gap.

2.2.3. Recurrent Neural Networks

RNNs [102] are designed to model sequential data by maintaining a hidden state that evolves over time. Unlike FNNs, which process each input independently, RNNs incorporate feedback connections that allow information to persist across time steps. This makes them suitable for tasks such as speech recognition, time series forecasting, and natural language modeling [63].

At each time step t, an RNN processes an input x_t and updates its hidden state h_t:

$$h_t = \phi(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

where $\phi(\cdot)$ is a nonlinear activation function. Figure 2.11 shows three common recurrent architectures: the vanilla RNN, the Long Short-Term Memory (LSTM) [103], and the Gated Recurrent Unit (GRU) [104].

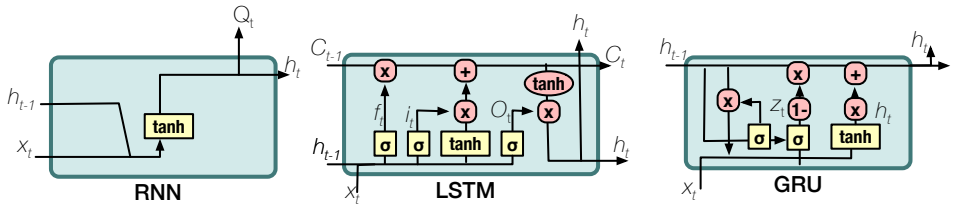

Figure 2.11.: Recurrent architectures: vanilla RNN, LSTM with gated memory cell, and GRU with update/reset gates [53].

While vanilla RNNs can in principle capture long-range dependencies, they are prone to vanishing or exploding gradients during training [105]. LSTMs and GRUs address this by introducing gating mechanisms that regulate the flow of information and gradients through time. LSTMs use separate input, output, and forget gates to control a persistent memory cell, while GRUs simplify this design with a single combined update gate. These mechanisms enable learning of long-term dependencies, though recurrent architectures remain inherently sequential, limiting their parallelization. Transformer models overcome this limitation by eliminating recurrence entirely.

2.2.4. Transformer Models

Transformers [106] represent a paradigm shift in sequence modeling. Instead of recurrence, they use self-attention mechanisms that directly relate all elements in a sequence to each other, enabling efficient modeling of long-range dependencies and full parallelization during training.

The core operation is scaled dot-product attention. Given queries Q, keys K, and values V:

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

where d_k is the key dimension and $K^\top$ denotes the transpose of the key matrix. Multi-head self-attention repeats this process in parallel with different learned projections, allowing the model to capture diverse relationships between tokens.

Transformers process an input sequence by first embedding the tokens and adding positional encodings to retain order information. These representations are then passed through stacked layers, each containing a multi-head attention module and a position-wise feedforward network [63]. Although originally developed for Natural Language Processing (NLP), Transformers have since been adapted to other domains. In Computer Vision, the Vision Transformer (ViT) [107] showed that a pure Transformer architecture can achieve state-of-the-art accuracy on image classification by treating an image as a sequence of patches. Unlike CNNs, which assume that nearby pixels are strongly related and that the same visual features can appear anywhere in an image, ViT makes no such built-in assumptions. Instead, it learns spatial relationships entirely from data through large-scale pretraining, and has inspired a wide family of variants for diverse tasks.

In NLP, Transformers underpin many of the most widely used large-scale pretrained language models. Generative Pretrained Transformer 2 (GPT-2) [108] is a decoder-only Transformer trained on massive text corpora with an autoregressive objective, predicting each next token given all previous ones. This simple setup enables it to generate coherent free-form text and to perform a broad range of tasks in a zero-shot setting, without any task-specific fine-tuning. Text-to-Text Transfer Transformer (T5) [109] takes a complementary approach: it uses an encoder-decoder Transformer and casts every problem (translation, classification, question answering) into a unified text-to-text format. This design allows knowledge learned from one task to transfer to others and has set new performance benchmarks across many standard NLP datasets.

Together, these advances illustrate how the Transformer architecture, initially designed for sequence modeling, has evolved into a general foundation for state-of-the-art models across both language and vision domains.

Chapter 3.

Blockchain, Cryptocoins, and Smart Contracts

This chapter introduces the foundational principles of blockchain systems that underpin decentralized cryptocoins and programmable platforms such as Bitcoin and Ethereum. Blockchains are compelling because they enable trustless coordination among mutually untrusted participants, eliminating the need for centralized authorities to enforce correctness. The chapter first examines the architecture of blockchains, covering their data structures and consensus mechanisms. It then discusses the economic layer of cryptocoins and token ecosystems, before concluding with an exploration of Smart Contracts and their role in enabling Decentralized Applications.

Chapter Outline

3.1. Blockchain Architecture and Mechanisms

At their core, blockchains [17] are append-only ledgers maintained by decentralized networks. Conceptually, a blockchain can be formalized as an ordered sequence of blocks

$$\mathscr{B} = (b_0, b_1, b_2, \dots)$$

where each block b_i contains a set of k_i transactions $\{t_{i,1}, \dots, t_{i,k_i}\}$ and a header referencing its parent via a cryptographic hash $h(b_{i-1})$. Here, k_i denotes the number of transactions in block i. The hash of each block depends on the hash of its predecessor, creating a back-linked chain of blocks extending to the genesis block b_0. Because changing any transaction would change its block's hash and invalidate all descendants, altering the deep history of the chain would require recomputing an infeasible amount of work [110, 111].

In Bitcoin [17, 110], this chain is often visualized as a vertical stack, where the block height denotes its distance from the genesis block and the tip denotes the most recent block. Temporary forks can occur when two miners produce blocks at nearly the same time, but the network eventually resolves these forks by adopting the valid chain with the greatest accumulated weight, according to the network's consensus rules, and discarding competing branches.

While this linked-block structure forms the foundation of all blockchain systems, platforms like Ethereum [18, 111] extend it with additional layers of functionality. Ethereum can be described as a deterministic state machine $\mathscr{S}$ operated by a decentralized peer-to-peer network. All nodes maintain a shared global state $s \in \mathscr{S}$, and the Ethereum Virtual Machine (EVM) applies transactions τ as state transitions

$$s_{t+1} = \delta(s_t, \tau_t)$$

where δ is the state transition function and t denotes a discrete time index (not to be confused with block height). The blockchain records not only the transactions themselves but also the resulting state updates. This design allows Ethereum to serve as a general-purpose decentralized computing platform, rather than solely as a payment network. Its native currency, Ether, functions primarily as a utility token to pay for computational resources and prevent abuse of the system.

More generally, most public blockchains share a common set of architectural components [110, 111]:

- a peer-to-peer network that propagates transactions and blocks,
- a set of consensus rules defining valid transactions and state transitions,
- a state machine that applies those transactions according to the rules,
- a chain of cryptographically linked blocks recording the history of state updates,
- a consensus algorithm that decentralizes control and enforces the rules,
- and an economic incentive mechanism (*e.g.*, block rewards or staking) that secures the network.

Together, these components constitute the layered architecture of a blockchain system. The following paragraphs focus on two of its most fundamental layers: the data structures that preserve an immutable and verifiable transaction history, and the consensus protocols that enable decentralized agreement on that history.

3.1.1. Data Structures

The fundamental data structure in a blockchain is the block. A block b_i aggregates a set of k_i transactions $\{t_{i,1}, \ldots, t_{i,k_i}\}$ alongside a fixed-size header containing metadata about the block itself and a link to its parent. Transactions are the atomic state transition operations of the system: in Bitcoin, they transfer ownership of coins between addresses, while in Ethereum they may additionally invoke Smart Contract code to modify on-chain state [17, 18].

Each transaction t is digitally signed by its creator using asymmetric cryptography, ensuring authenticity and non-repudiation. Transactions are identified by a cryptographic hash $\mathsf{txid}(t)$ and are immutable: any modification to their contents changes their hash. Blocks group these signed transactions into an ordered batch, providing a shared and append-only timeline of state updates across the network.

A complete block consists of two main components [110]:

- the block header, a small fixed-size structure containing summary information about the block (such as its position in the chain, creation time, and a unique identifier),
- the transaction list, a variable-length sequence of the transactions included in the block, which typically accounts for almost all of its size.

Blocks are identified by the cryptographic hash of their header:

$$\mathsf{hash}(b_i) = h(\mathsf{header}(b_i))$$

where $h(\cdot)$ denotes the network's chosen hash function (SHA-256d in Bitcoin [17], Keccak-256 in Ethereum [18]). Because the header includes the hash of its parent $h(b_{i-1})$, any change to an ancestor propagates forward, altering all its descendants. This property underlies the blockchain's immutability: once a block has accumulated sufficient successors, rewriting it would require recomputing all subsequent blocks, which is computationally infeasible in Proof-of-Work (PoW) systems [112] (see Section 3.1.2).

In addition to the block hash, blocks are also referred to by their height, defined as their distance from the genesis block. The height gives their position in the chain but is not unique, temporary forks can produce multiple competing blocks at the same height. The hash, by contrast, uniquely identifies a block.

Merkle Trees

To efficiently commit to and verify the potentially large set of transactions within a block, most blockchains use a Merkle tree [113]. A Merkle tree is a binary hash tree constructed bottom-up: pairs of transaction hashes are concatenated and hashed to form parent nodes, which are then paired and hashed again, until a single 32-byte Merkle root remains.

Formally, given transactions $(t_1, \ldots, t_n)$ and a cryptographic hash $h(\cdot)$, first define the leaves as

$$l_i = h(t_i) \qquad (i = 1, \ldots, n)$$

and then the root as

$$\mathrm{root}(t_1, \ldots, t_n) = h\big(h(l_1 \| l_2) \parallel h(l_3 \| l_4) \parallel \ldots\big).$$

If the number of leaves is odd, the final hash is duplicated to ensure an even number of children at each level. The resulting root is stored in the block header and serves as a compact commitment to the entire set of transactions.

A crucial property of Merkle trees is that verifying the inclusion of a single transaction requires only a logarithmic number of hashes along its path to the root. This allows Simplified Payment Verifications (SPVs) clients to verify that a transaction is included in a block without downloading the entire block, by requesting only the block header and a short Merkle proof [110]. Figure 3.1 shows an example with four transactions: their leaf hashes are combined pairwise, then those pairs are combined again to produce the final Merkle root.

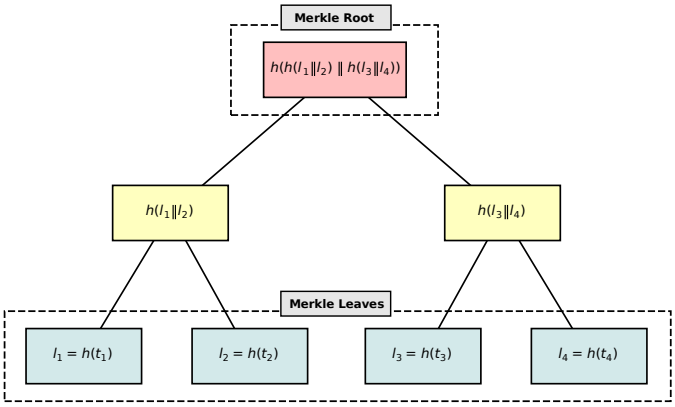

Figure 3.1.: Merkle tree of four transactions $t_1, \ldots, t_4$, showing their leaf hashes $l_i = h(t_i)$ and the resulting Merkle root $h(h(l_1 \| l_2) \| h(l_3 \| l_4))$.

Block Linking and State Tries

Each block header contains the hash of its parent block. This creates a back-linked chain:

$$\cdots \xrightarrow{h(b_{i-2})} b_{i-1} \xrightarrow{h(b_{i-1})} b_i$$

where altering any block would invalidate all its descendants. Nodes use this linkage to continuously extend their local copy of the chain, always appending new blocks to the current tip with the highest cumulative work (or stake). Because the hash of a block depends on its parent, the chain forms a tamper-evident history: changing a single transaction would require recomputing the proof for that block and all subsequent ones [110].

Ethereum follows the same structural principle but adds an additional layer: while Bitcoin only stores transactions in a Merkle tree, Ethereum stores the entire system state (accounts, balances, contract storage) in a Merkle Patricia Trie (MPT) [18]. The MPT combines the properties of a Merkle tree and a Patricia trie [114]: it is a cryptographically authenticated, prefix-compressed key-value store in which every modification changes the root hash. This structure allows Ethereum to store and verify its global state efficiently, while enabling light clients to check state proofs with logarithmic complexity. Each Ethereum block header therefore commits not only to the transactions via a transaction trie root, but also to the post-execution global state root and the receipts root. This design enables Ethereum nodes to verify both transaction inclusion and the resulting state transitions without maintaining full historical data [111].

Together, these data structures (transaction lists, Merkle trees, block headers, and inter-block hashes) form the blockchain's core integrity layer. They ensure that all participants converge on a consistent append-only history, providing the foundation on which consensus protocols and decentralized applications are built.

3.1.2. Consensus Protocols

Maintaining a consistent shared history among mutually untrusted participants in a decentralized network is a fundamental challenge, often framed as the Byzantine Generals Problem [115]. In this classic thought experiment, a group of generals must coordinate a joint attack or retreat, even though some may be traitors who send conflicting or false messages. Achieving consensus requires that all honest generals agree on a single plan despite the presence of Byzantine faults.

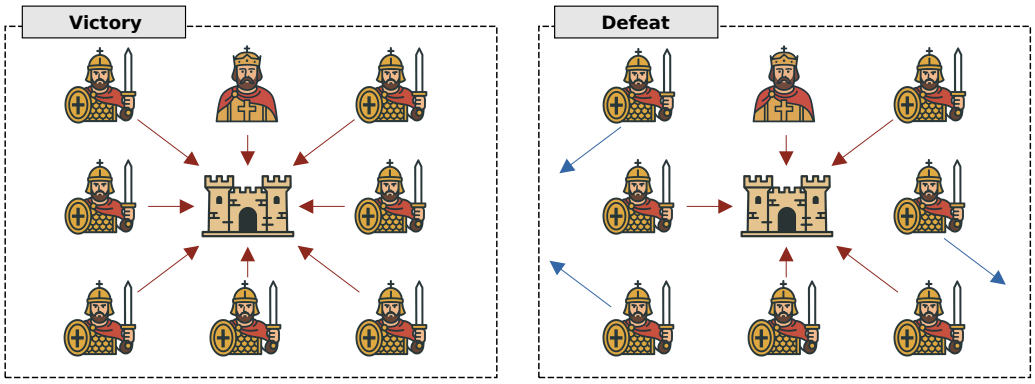

Figure 3.2.: The Byzantine Generals Problem. Left: all generals agree to attack, achieving consensus and victory. Right: inconsistent decisions lead to failure.

As illustrated in Figure 3.2, when all participants relay the same decision faithfully, consensus emerges and the coordinated plan succeeds. If even a subset acts inconsistently, however, the group fails. Blockchains solve this dilemma in open networks (where any node may join and potentially act maliciously) through consensus protocols that align incentives toward correct behavior and make deviation economically or computationally prohibitive.

Bitcoin addresses the Byzantine Generals Problem using Proof-of-Work (PoW) [17]. In PoW, miners compete to solve a computational puzzle: they repeatedly hash candidate block headers while varying a nonce until producing an output below a network-wide difficulty target. This process is computationally expensive but trivial for others to verify. The first miner to find a valid solution broadcasts their block; if the block's contents satisfy the consensus rules, the network accepts it and extends the chain. The canonical chain is defined as the one with the greatest cumulative PoW (measured from each block's target difficulty) [110, 112]. Rewriting history would require redoing all the PoW on the replaced blocks and catching up to the honest chain, which would demand controlling more than half of the network's total hash rate. PoW provides probabilistic finality: as additional blocks build on top of a given block, the cost and probability of successfully rewriting it drop exponentially, making deep reorganizations practically impossible. Bitcoin's security ultimately stems from the economic cost of computation, with block rewards and transaction fees incentivizing honest participation and disincentivizing attacks [110, 112].

Ethereum originally employed a similar PoW structure, but using the Ethash algorithm, which was designed to be memory-hard to reduce the advantage of specialized hardware [18]. However, Ethereum has since transitioned to Proof-of-Stake (PoS) [116] through the Ethereum 2.0 (Merge) upgrade, replacing energy-intensive mining with validator-based consensus [18, 111]. In PoS, participants lock up a stake of the native currency as collateral to become validators. Time is divided into slots and epochs; in each slot, a randomly selected validator proposes a block, and others attest to its validity. Finality is achieved through the Casper Friendly Finality Gadget (FFG) overlay protocol [117]: once two-thirds of the staked validators attest to a checkpoint, it becomes finalized, and reverting it would require violating the protocol in a way that provably destroys at least one-third of the total staked funds through slashing.

Both PoW and PoS operate in open, adversarial networks and aim to achieve consensus under eventual synchrony. PoW secures the chain by making history expensive to rewrite, while PoS secures it by making dishonest behavior economically self-destructive through the risk of losing staked collateral [110, 111]. These consensus protocols form the trustless backbone of blockchain networks, enabling the secure issuance and transfer of cryptocoins that underpin their economic layer.

3.2. Cryptocoins and Token Economies

The consensus protocols described in Section 3.1.2 secure a blockchain's shared ledger, enabling the issuance and transfer of native digital assets. These assets, commonly referred to as cryptocoins, constitute the economic layer that incentivizes network participation and powers decentralized ecosystems. Whereas the architectural layer ensures the integrity and availability

of state, the economic layer governs the creation, exchange, and distribution of value among participants. This section first surveys major cryptocoins and their derivatives, and then examines the dynamics and challenges of their markets.

3.2.1. Major Cryptocoins and Altcoins

Bitcoin (BTC) [17] was the first operational cryptocoin, introduced in 2009 by the pseudonymous Satoshi Nakamoto as a decentralized peer-to-peer payment system. BTC pioneered the notion of a scarce digital asset secured purely through cryptographic proof and decentralized consensus (PoW). The first recorded real-world BTC transaction occurred at block #170, when Nakamoto transferred coins to another participant [118], demonstrating its potential as a form of digital cash.

Ethereum's native currency, Ether (ETH) [18], extended the concept of cryptocoins beyond simple payment functionality. ETH functions as a utility token that fuels computation in the EVM, compensating miners (or validators under PoS) for executing transactions and smart contracts. While BTC is deliberately conservative in design, emphasizing security and monetary soundness, Ethereum's programmable architecture enables a broad ecosystem of Decentralized Applications (dApps), Decentralized Finance (DeFi) protocols, and token issuance frameworks (notably ERC-20 tokens [119]).

Beyond these two maincoins, the market encompasses thousands of alternative coins (altcoins). Prominent examples include:

- Litecoin (LTC) [120], a Bitcoin fork with shorter block times and a Scrypt-based hashing algorithm,
- Cardano (ADA) [121], a PoS-based blockchain emphasizing formal verification and sustainability,
- Ripple (XRP) [122], designed for cross-border payments using a federated consensus model rather than traditional mining,
- Binance Coin (BNB) [123], initially issued by the Binance exchange for fee payments and now used to secure the BNB Chain.

A distinct subclass are stablecoins, which aim to mitigate volatility by pegging their value to external reference assets. Examples include Tether (USDT) [124] and USD Coin (USDC) [125], both of which maintain parity with the U.S. dollar by being backed with reserves. Unlike free-floating cryptocoins such as BTC or ETH, stablecoins are designed to provide a predictable unit of account and store of value, making them foundational components of DeFi protocols and crypto-based financial infrastructure.

3.2.2. Market Dynamics and Challenges

The cryptocoin market has evolved into a large-scale global economy. As of late 2022, over 9,000 coins were listed on public market trackers such as CoinMarketCap, with an aggregate market capitalization exceeding 1.7 trillion USD, comparable to the GDP of South Korea in 2021 [53, 126]. Cryptocoins are primarily traded on online exchanges, either centralized (CEX) platforms such as Coinbase [127], Kraken [128], or Binance [129], or decentralized (DEX) platforms such as Uniswap [130]. These exchanges operate continuous order books and liquidity pools, enabling high-frequency and around-the-clock trading.

A defining characteristic of cryptocoin markets is their pronounced price volatility. Historical price series exhibit sharp fluctuations over short time horizons, with abrupt surges and crashes that exceed those typical of traditional financial assets. Figure 3.3 shows the normalized daily OHLC (open-high-low-close) prices of the five most-traded coins (BTC, ETH, BNB, XRP, ADA) between March 2020 and March 2023, highlighting periods of strong upward and downward trends. Within a single six-month window (*e.g.*, March to September 2021), most major assets experienced price swings exceeding an order of magnitude. Such behavior has been empirically shown to exhibit time-varying and clustered volatility [22], where large price moves tend to be followed by further large moves, contradicting the random walk hypothesis [131].

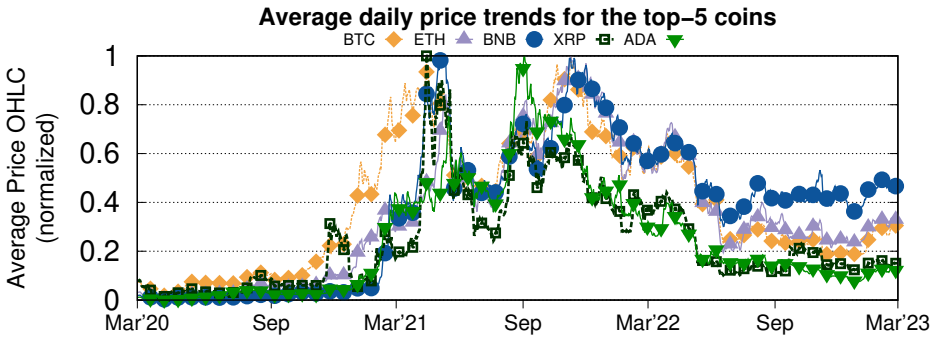

Figure 3.3.: Average normalized OHLC daily prices for BTC, ETH, BNB, XRP, and ADA from March 2020 to March 2023 [53].

Several structural factors contribute to this volatility [53]:

- the speculative nature of crypto-asset investment and the absence of intrinsic valuation models,
- limited regulatory oversight or government-backed guarantees,
- concentration of ownership (whales) capable of triggering large market swings through coordinated buy/sell actions (pump-and-dump dynamics),
- and the sensitivity of market sentiment to social media activity, news cycles, and macroeconomic events.

In addition to volatility, price correlations between cryptocoins are often observed, leading to synchronized price movements during bullish or bearish phases. For instance, BTC and ETH frequently exhibit strong co-movement, with many altcoins following similar trends. Major exchanges such as Coinbase now publish correlation indices based on rolling Pearson correlation coefficients computed from USD-denominated order books over 90-day windows [132]. These metrics are presented to end-users through mobile trading interfaces, potentially shaping their investment strategies. However, the structure and evolution of such correlations remain largely unexplored. Moreover, exchanges typically expose only a limited and curated subset of correlation indicators. In our own studies [52, 53], we analyzed these correlation patterns in depth and uncovered several underlying characteristics; the results of this analysis will be presented in Chapter 5.

Overall, cryptocoin markets operate at the intersection of technological innovation and speculative finance. Their decentralized nature, coupled with global accessibility and the absence of central authorities, fuels both their rapid growth and their instability. Understanding their market dynamics is crucial for designing secure token economies, developing robust trading strategies, and informing regulatory frameworks that protect end-users without stifling innovation.

The emergence of programmable platforms such as Ethereum has expanded the role of cryptocoins from simple value-transfer instruments to enablers of complex on-chain economies. The next section examines Smart Contracts, the key mechanism that allows developers to encode business logic directly on the blockchain, automate financial interactions, and build decentralized applications that leverage the token economies introduced here.

3.3. Smart Contracts and Decentralized Applications

Smart Contracts (SCs) extend blockchains beyond simple peer-to-peer payment systems into fully programmable platforms capable of hosting dApps. The concept was first articulated by Szabo in 1997 as self-executing agreements that enforce contractual clauses through code rather than legal institutions [19]. With the advent of platforms such as Ethereum, this vision became technically feasible: Ethereum integrates a Turing-complete execution environment into its consensus layer, enabling arbitrary computation to be embedded within the blockchain [18]. An SC is thus an autonomous program deployed on-chain. Once published, its bytecode and persistent state are immutably recorded in the global ledger, while its public functions can be invoked through transactions. This paradigm supports a wide spectrum of applications, from financial instruments and decentralized exchanges to digital art marketplaces, games, and governance protocols, without reliance on centralized intermediaries.

dApps are decentralized applications that employ one or more SCs as their core execution logic, typically complemented by off-chain components such as user interfaces, oracles, and auxiliary storage systems. In contrast to conventional applications that rely on centralized servers and administrators, dApps execute directly on blockchain networks, thereby inheriting their transparency, immutability, and resistance to censorship [111]. This architectural shift enables the

Table 3.1.: Subset of EVM opcodes as of the Shanghai fork [137].

Opcode	Name	Gas	Description
0x00	STOP	0	Halts execution
0x01	ADD	3	Addition operation
0x02	MUL	5	Multiplication operation
...	...	...	...
0xFD	REVERT	0	Halts reverting state changes
0xFE	INVALID	NaN	Designated invalid instruction
0xFF	SELFDESTRUCT	5000	Halts deleting account

creation of financial services, decentralized exchanges, and governance frameworks that function without trusted intermediaries, laying the foundation for the rapidly growing ecosystem of DeFi.

In practice, dApps consist of on-chain contracts that are accessed through user wallets and off-chain clients (typically web or mobile interfaces) that mediate interaction with the blockchain. Because every contract invocation is executed and verified across all participating nodes, correctness and determinism are enforced by consensus, ensuring that all users observe an identical outcome. Yet, the same immutability that guarantees transparency and trustlessness also renders deployed contracts difficult to modify, amplifying the consequences of programming errors or security vulnerabilities [133, 134]. This tension between reliability and adaptability has become a central challenge in the design and governance of SCs.

3.3.1. Core Principles and Execution Environments

Ethereum pioneered the execution of general-purpose SCs through the EVM, a stack-based, 256-bit virtual machine that interprets low-level bytecode [18]. High-level languages such as Solidity [135] and Vyper [136] compile into this bytecode, which exposes opcodes for arithmetic (`ADD`, `SUB`), signed math (`SDIV`, `SMOD`), hashing (`SHA3`), memory and stack manipulation, and inter-contract execution (`CALL`, `DELEGATECALL`). Contracts themselves can be instantiated (`CREATE`, `CREATE2`) or terminated (`SELFDESTRUCT`) via dedicated instructions. To prevent denial-of-service attacks and promote economic sustainability, execution is metered in gas, a unit of computation that must be paid in the native currency. Each transaction therefore represents a state transition in Ethereum's global state machine, with gas fees compensating validators for the resources consumed. Table 3.1 presents a representative subset of EVM opcodes as of the Shanghai fork [137], highlighting the diversity of operations from arithmetic to contract lifecycle management.

The EVM was designed with portability and verifiability in mind: every node in the network executes the same bytecode deterministically, allowing independent verification of results. Yet, bytecode-level execution renders contract logic opaque to human readers, and the growing complexity of the opcode set (144 distinct instructions as of the Shanghai upgrade [137]) makes auditing and static analysis increasingly difficult. These challenges have spurred the development of dedicated analysis frameworks, including symbolic execution engines, formal

verification methods, and more recently, ML-based techniques for detecting vulnerabilities and malicious behavior [138–148].

Beyond Ethereum, several blockchain platforms have introduced alternative smart contract execution environments. A prominent trend is the adoption of WebAssembly (WASM) as a contract runtime by networks such as Polkadot [149], NEAR [150], the Internet Computer [151], and EOSIO [152]. WASM offers platform independence, near-native execution performance, and compatibility with a wide range of programming languages through LLVM [153]. In contrast to the EVM's specialized stack machine, WASM is a general-purpose bytecode format featuring a sandboxed runtime and a standardized system interface (WASI). Reflecting this shift, Ethereum itself is exploring a transition to eWASM [154], underscoring the growing role of WASM as a unifying layer for decentralized computation.

Both the EVM and WASM execution environments share several core principles: deterministic computation replicated across all nodes, bounded resource usage through gas or metering, and transparent on-chain persistence of code and state. Their design philosophies, however, diverge. The EVM was purpose-built for Ethereum, reflecting its specific requirements and historical constraints, whereas WASM is a modular and portable architecture designed for general-purpose computation and increasingly adopted by newer platforms. While these environments provide the substrate on which SCs operate, they also broaden the attack surface, introducing new vectors for adversaries to exploit [25, 26, 155].

3.3.2. Security Challenges in Smart Contracts

The immutability and high financial stakes of SCs make their security a critical concern. Once deployed, a contract's code is effectively immutable: while upgrade mechanisms such as proxy patterns exist [133], they are complex to implement and can introduce new risks [134]. As a result, any flaw may persist indefinitely as a potential exploit vector. This threat is not merely theoretical: attacks on Ethereum SCs have repeatedly resulted in losses of hundreds of millions of USD. Notable examples include the 2016 Decentralized Autonomous Organization (DAO) reentrancy exploit [156] and the 2021 Grim Finance attack [157]. The scale of the problem continues to grow: reported losses from Ethereum SC breaches exceeded 1.3 billion USD in 2023 alone [158].

Beyond such vulnerabilities, adversaries also target the human element through phishing and malware contracts, which often masquerade as legitimate dApps. In the blockchain context, phishing denotes social engineering attacks in which users are deceived into authorizing malicious transactions. Common tactics include distributing links to fraudulent applications, luring victims with fake incentives, or publishing typosquatted contract addresses that closely resemble genuine ones [55, 56, 159, 160]. Once a user approves a seemingly innocuous action, the contract's hidden logic can expropriate their funds. To mitigate these threats, static detection frameworks analyze bytecode patterns prior to deployment, offering early warnings without relying on user interaction data. Adversaries, however, increasingly employ obfuscation techniques, either at the source-code level [31] or directly in bytecode [32–34], that rewrite control flow and instruction layout while preserving malicious intent, thereby evading detection.

The diversity of execution environments further complicates smart contract security. Most existing analysis tools are tailored to EVM bytecode, yet the growing adoption of WASM-based platforms introduces heterogeneous instruction sets and distinct runtime semantics. Early efforts such as EOSAFE [161], WASMOD [162] and WASAI [163] address structural vulnerabilities such as unchecked calls and arithmetic overflows, but comprehensive frameworks capable of detecting malicious logic in WASM SCs remain limited. As blockchain ecosystems diversify, the need for cross-platform detection methods that combine program analysis with ML techniques is becoming increasingly urgent.

In summary, SCs transform blockchains from simple value-transfer systems into programmable platforms that drive dApps and decentralized economies. By embedding logic directly on-chain, they enable trustless automation but also introduce new security risks. Mitigating these risks demands cross-platform detection techniques that can identify malicious behavior in both EVM and WASM-based environments while remaining robust against adversarial obfuscation. Combined with the architectural foundations of blockchains and the economic dynamics of cryptocoins, this security dimension forms the substrate on which the broader landscape of decentralized applications is built.

Part II.

Strategies for Efficient Machine Learning Inference

Chapter 4.

Model-Serving Frameworks: an Experimental Evaluation

In this chapter, we present a systematic evaluation of model-serving frameworks for deploying Machine Learning models in production environments. Serving strategies are critical because inference is the stage where end-users directly interact with predictive systems, making efficiency, scalability, and reliability essential for practical adoption. We compare five representative open-source frameworks across diverse workloads, analyzing the trade-offs between domain-specific solutions tailored to Deep Learning and more general-purpose approaches. The findings of this evaluation establish a foundation for the subsequent contributions on forecasting in Decentralized Finance and security of Smart Contracts, where efficient and trustworthy inference is a prerequisite for real-world deployment.

Chapter Outline

4.1. Introduction

Deploying Machine Learning (ML) models into production requires not only accurate algorithms but also efficient serving strategies that deliver predictions reliably at scale. While training is computationally intensive and often performed offline on specialized hardware [35, 36], inference is the stage where end-users interact with predictive systems. Here, latency, throughput, and cost constraints become decisive: forecasts lose value if delivered after market shifts, and delayed detection can undermine security guarantees [37].

Commercial Machine Learning as a Service (MLaaS) offerings [38] from major providers [39, 40] address this need by exposing trained models through scalable, cloud-hosted services. These platforms provide seamless integration and elasticity but come with high economic costs [41–43] and raise concerns about privacy, control, and trust [44, 45].

An alternative is the use of open-source serving frameworks, which allow organizations to containerize and deploy models on-premises or in custom environments. Such frameworks support scalable inference via public Application Programming Interfaces (APIs), often at a fraction of the operational cost of commercial solutions. Despite their importance, a systematic and reproducible evaluation of these frameworks remains limited.

In this chapter, we address this gap through a comparative study of five widely used open-source model-serving frameworks: TensorFlow Serving [46], TorchServe [47], MLServer [50], MLflow [49], and BentoML [48]. The selection covers both Deep Learning (DL)-specific frameworks (TensorFlow Serving and TorchServe), which are the de facto standards for TensorFlow [164] and PyTorch [165] models, and general-purpose frameworks (MLServer, MLflow, BentoML) that support heterogeneous ML workloads.

We benchmark these frameworks across four representative application domains (malware detection, cryptocoin forecasting, image classification, and sentiment analysis) capturing a spectrum of real-world workloads. Our evaluation highlights the trade-offs between domain-specific and general-purpose solutions, providing practical guidance for deploying ML inference services at scale.

The contributions of this study [51] are threefold:

- A systematic analysis of five open-source model-serving frameworks under four representative workloads;
- An empirical comparison of inference performance, identifying TensorFlow Serving as the fastest solution for DL workloads;
- An assessment of trade-offs between specialized and general-purpose frameworks, informing the design of efficient and reproducible deployment pipelines.

All code, datasets, and pre-trained models are released as open-source artifacts to ensure reproducibility and facilitate further research [60].

4.2. Related Work

Research on model-serving strategies is situated within the broader field of Machine Learning Operations (MLOps). MLOps is commonly defined as a paradigm encompassing best practices, conceptual frameworks, and development culture for the end-to-end design, implementation, deployment, monitoring, and scalability of ML systems [166]. It integrates principles from ML, software engineering, and data engineering, with the overarching goal of bridging the gap between development and operations. Within this context, model serving constitutes a critical component, as it directly affects the reliability, performance, and scalability of ML products in production environments.

Early studies investigated the integration of serving mechanisms into data processing frameworks. For example, Horchidan et al. [167] evaluated pipelines built on Apache Flink for image classification tasks. Their results showed that embedding pre-trained models directly into the stream processing logic improved throughput. When comparing TensorFlow Serving and TorchServe, they consistently observed superior performance for TensorFlow Serving, a finding that resonates with our own evaluation.

More recently, attention has shifted toward the specialized challenges of serving large-scale Large Language Models (LLMs). Miao et al. [168] introduced SpecInfer, a system that reduces generative inference latency through tree-based speculative decoding with auxiliary models. Strati et al. [169] proposed Déjà Vu, which optimizes distributed LLM serving by introducing prompt-token disaggregation, micro-batch swapping for improved Graphics Processing Unit (GPU) utilization, and state replication for fault tolerance. Complementing these system designs, [170] released the first real-world trace dataset of LLM serving workloads, enabling systematic benchmarking of serving techniques under realistic conditions.

Together, these contributions illustrate the rapid evolution of serving research: from general-purpose pipelines toward highly specialized systems tailored to LLMs. Our study complements this trajectory by shifting the focus back to open-source serving frameworks for a broader range of workloads, providing a reproducible comparative analysis that extends beyond the current emphasis on LLMs.

4.3. Model-Serving Frameworks

This section reviews the five model-serving frameworks included in our evaluation: TensorFlow Serving, TorchServe, MLServer, MLflow, and BentoML. Their main characteristics, ranging from hardware support to compatibility with ML/DL libraries and model formats, are summarized in Table 4.1.

TensorFlow Serving

TensorFlow Serving [46] is a high-performance system for deploying TensorFlow models in production. Models are stored in the `SavedModel` format, which encapsulates variables, pa-

Table 4.1.: Overview of the model-serving frameworks considered in this study (supported ML/DL libraries and model formats are based on official documentation).

Framework	ML	DL	CPU	GPU	Supported (ML)	Supported (DL)	Model Formats
TensorFlow Serving	✗	✓	✓	✓	-	TensorFlow, HuggingFace, Keras	SavedModel, TFLite, TF.js
TorchServe	✗	✓	✓	✓	-	PyTorch, HuggingFace	MAR, PyTorch, ONNX
MLServer	✓	✓	✓	✓	Scikit-Learn, XGBoost, LightGBM, CatBoost	TensorFlow, PyTorch, HuggingFace, Keras	SavedModel, PyTorch, Keras, ONNX, Pickle, Joblib
MLflow	✓	✓	✓	✓	Scikit-Learn, XGBoost, LightGBM, CatBoost	TensorFlow, PyTorch, HuggingFace, Keras	SavedModel, PyTorch, Keras, ONNX, Pickle, Joblib
BentoML	✓	✓	✓	✓	Scikit-Learn, XGBoost, LightGBM, CatBoost	TensorFlow, PyTorch, HuggingFace, Keras	SavedModel, PyTorch, Keras, ONNX, Pickle, Joblib

rameters, and computation graphs, and can then be exposed via Representational State Transfer (REST) or Google Remote Procedure Call (gRPC) interfaces. While tightly coupled to the TensorFlow ecosystem, the framework also supports Keras [171] and HuggingFace [172] models, making it the de facto standard for TensorFlow-based deployments.

TorchServe

TorchServe [47] extends PyTorch with native serving capabilities. Trained models, saved as `.pth` or `.pt`, are packaged into a Model Archive (MAR) using the Torch Model Archiver [173]. TorchServe supports TorchScript [174] and Open Neural Network Exchange (ONNX) [175] models, and allows simultaneous serving of multiple models within one instance. It further enables customization through user-defined handlers for preprocessing and postprocessing, which has contributed to its popularity for serving HuggingFace Transformers.

MLServer

MLServer [50] is a lightweight framework designed for heterogeneous ML workloads. In addition to TensorFlow and PyTorch, it natively supports Scikit-Learn [176], XGBoost [89], LightGBM [91], and CatBoost [92]. MLServer exposes inference through REST endpoints and provides runtime environments tailored to specific model requirements, balancing flexibility with scalability.

MLflow

MLflow [49] offers a broader platform for the ML lifecycle, spanning training, experiment tracking, and deployment. Beyond logging and reproducibility features, MLflow supports serving through Flask [177] or external backends such as MLServer. Its versatility makes it particularly suited for projects requiring integrated experiment management and deployment pipelines.

BentoML

BentoML [48] provides a unified model-serving solution emphasizing reproducibility. Models are packaged into self-contained "Bentos", which include pre-trained weights, dependencies, and optional preprocessing or postprocessing steps. By bundling artifacts with their environment specifications, BentoML simplifies scalable deployments and facilitates portability across infrastructures.

4.3.1. Model Formats and Conversions

As Table 4.1 illustrates, framework compatibility strongly depends on the supported model formats. TensorFlow Serving and TorchServe are optimized for their respective ecosystems, whereas MLServer, MLflow, and BentoML adopt a general-purpose design that covers a wider set of libraries.

This diversity introduces challenges for cross-framework comparisons. For example, TorchServe cannot natively handle TensorFlow's `SavedModel` format. Achieving parity requires converting models into PyTorch-compatible formats (`.pth/.pt`). The most widely adopted solution for such interoperability is ONNX, which provides a standard representation for DL models. Models trained in TensorFlow can be exported to ONNX and subsequently imported into PyTorch. While this enables comparative benchmarking, it may also affect fidelity due to subtle differences in execution semantics across frameworks, including variations in operator implementations, numerical precision, and the handling of framework-specific layers or optimizations during conversion.

To ensure fair evaluation, we employed two conversion libraries: TF2ONNX [178] for exporting TensorFlow models, and ONNX2Torch [179] for converting ONNX representations into PyTorch. This pipeline allowed us to systematically compare TensorFlow Serving and TorchServe, covering the two most widely adopted DL ecosystems.

4.4. Motivating Scenarios

To capture the diversity of real-world workloads, our evaluation considers four motivating scenarios spanning security, finance, vision, and language. Each scenario represents a distinct class of inference task, allowing us to assess serving frameworks under heterogeneous requirements.

We note that the models used in these scenarios are not optimized for state-of-the-art predictive performance; instead, they are trained using standard architectures and widely adopted datasets to produce representative inference workloads. This design ensures that our conclusions reflect realistic serving behavior, such as model complexity, input characteristics, and computational demands, rather than task-specific accuracy, which is orthogonal to the objectives of this study.

Malware detection on Android Packages (APKs) involves analyzing file features to determine whether an app exhibits malicious behavior. Modern approaches rely on ML and DL models to automate this process [180–183]. Malware analysis can be conducted through static methods, which extract features without execution, or dynamic methods, which monitor runtime behavior [180, 181]. In line with prior work showing the effectiveness of static features for ML-based detection [184], we adopt Drebin [183] to extract binary feature vectors from APKs. A simple Multi-Layer Perceptron (MLP) with two hidden layers and a sigmoid output is then trained on a 10% subsample of the CICMalDroid 2020 dataset [185] to classify applications as benign or malicious.

Cryptocoin forecasting aims to predict future price movements by leveraging correlations across multiple assets. Given the volatility of cryptocurrency markets, forecasting remains challenging yet highly relevant [53, 186, 187]. In this scenario, we focus on predicting hourly Bitcoin (BTC) prices using the closing values of eight correlated coins: Cardano (ADA), Binance Coin (BNB), Dogecoin (DOGE), Ether (ETH), Litecoin (LTC), Solana (SOL), Stellar Lumens (XLM), and Ripple (XRP). Inputs are smoothed with a 24-hour Moving Average (MA), and a two-layer MLP with Rectified Linear Unit (ReLU) activations is trained on data collected from Binance [129] covering January-April 2024 [60].

Image classification is a canonical task in Computer Vision with wide-ranging applications, including medical diagnosis [188] and object detection [189]. For this scenario, we adopt MobileNetV2 [190], a lightweight Convolutional Neural Network (CNN) architecture based on inverted residuals. The model is pre-trained on the ImageNet 2012 dataset [191], which comprises over 1.2 million images across 1,000 classes, and configured for input dimensions of $224 \times 224 \times 3$.

Sentiment analysis (opinion mining) involves detecting emotions, opinions, or attitudes expressed in text, with applications in social media monitoring [192] and customer feedback analysis [193]. We employ a CNN architecture with an embedding layer, two convolutional layers, and a sigmoid output for binary sentiment classification. Input text undergoes tokenization, stopword and punctuation removal, and lemmatization before training. The model is trained on the IMDb 50K dataset [194], containing 50,000 labeled movie reviews.

Table 4.2 provides a concise overview of the datasets and models used in each scenario, highlighting the diversity of tasks considered in our evaluation.

Table 4.2.: Motivating scenarios with corresponding datasets and models.

Use Case	Dataset	Model
Malware Detection	CICMalDroid 2020 (10%) [185]	MLP + Drebin [183]
Cryptocoin Forecasting	Binance (Jan-Apr 2024) [60]	MLP + Moving Average
Image Classification	ImageNet 2012 [191]	MobileNetV2 [190]
Sentiment Analysis	IMDb 50K [194]	Embedding + CNN

4.5. Datasets

A robust evaluation of serving frameworks requires datasets that are both representative of real-world applications and diverse across domains. To this end, we draw on four established benchmarks, each corresponding to one of the motivating scenarios described in Section 4.4: CICMalDroid 2020 [185], Binance [60], ImageNet [191], and IMDb 50K [194].

CICMalDroid 2020. CICMalDroid 2020 is an Android malware dataset comprising 17,341 applications collected between December 2017 and December 2018 from sources such as VirusTotal [195] and Contagio [196]. Samples are labeled into five categories: adware, banking malware, SMS malware, riskware, and benign. For our study, we select a representative 10% subsample (1,727 applications) while preserving the original class distribution.

Binance. The Binance dataset contains high-resolution time series data from the Binance exchange, recording hourly closing prices[1] for nine major cryptocurrencies: ADA, BNB, BTC, DOGE, ETH, LTC, SOL, XLM, and XRP. Covering January 1 to April 19, 2024, the dataset spans 23,544 hourly observations, enabling the analysis of correlations and co-movements across assets. This dataset was curated specifically for our study and is publicly released as part of our reproducibility package [60].

ImageNet. ImageNet is a large-scale benchmark for Computer Vision tasks, organized according to the WordNet hierarchy [197]. We rely on the ILSVRC 2012 subset [198], which contains 1.28 million training images, 50,000 validation images, and 100,000 test images distributed across 1,000 object classes. Despite its age, it remains a standard benchmark for image classification.

IMDb 50K. The IMDb 50K dataset consists of 50,000 highly polarized movie reviews annotated for binary sentiment (positive/negative). It is evenly split into training and test sets of 25,000 reviews each and is widely used for benchmarking sentiment analysis models in Natural Language Processing (NLP).

Together, these datasets provide a diverse foundation for evaluating serving frameworks under heterogeneous workloads, spanning security, finance, vision, and language.

[1]The final transaction price within a given hour.

4.6. Evaluation

This section presents a systematic experimental evaluation of the five model-serving frameworks considered in this study: TensorFlow Serving, TorchServe, MLServer, MLflow, and BentoML. The evaluation spans the four motivating scenarios introduced in Section 4.4, with the goal of assessing framework efficiency under realistic inference workloads. Our analysis focuses on inference latency, as this metric directly determines the responsiveness and usability of ML services in production.

The experimental setup is designed to reflect common real-world deployment practices for latency-sensitive ML services. Specifically, models are served via standard REST interfaces on CPU-based infrastructure, single-request inference is used (batch size one), and workloads are driven by real test data at sustained request rates. While large-scale production systems may incorporate GPUs, autoscaling, or multi-stage pipelines, isolating framework-level inference behavior under controlled conditions enables a fair and reproducible comparison of serving efficiency across frameworks.

To guide the study, we address the following research questions:

- **Q1:** Are there significant differences in observed latencies across frameworks?
- **Q2:** How does input payload size influence latency distributions?
- **Q3:** Which framework achieves the lowest latencies for DL workloads?
- **Q4:** Do DL-specific frameworks (TensorFlow Serving, TorchServe) consistently outperform general-purpose solutions (MLflow, MLServer, BentoML)?

By answering these questions, we aim to identify the most efficient serving strategies and clarify the trade-offs between specialized and general-purpose frameworks. All code, datasets, and replication instructions are publicly released to ensure transparency and reproducibility [60].

4.6.1. Experimental Setup

All experiments were conducted on a dedicated Ubuntu 22.04.4 LTS server (Linux kernel 5.15.0-105-generic), equipped with 40 Intel Xeon® Gold 5215 CPUs at 2.50 GHz and 128 GB RAM, with no GPU acceleration employed.

The software environment included Python 3.10.12 and the following framework versions: TensorFlow ModelServer 2.15.0, PyTorch 2.2.2 with TorchServe 0.10.0, BentoML 1.2.12, MLServer 1.5.0, and MLflow 2.12.1.

All models were trained in TensorFlow. Since TorchServe natively supports only PyTorch model formats (`.pth`), we converted models to PyTorch via the ONNX interoperability standard [175]. Specifically, TF2ONNX [178] (v1.16.1) was used to export TensorFlow models to ONNX (v1.16.0), and ONNX2Torch [179] (v1.5.14) to generate PyTorch-compatible artifacts.

Table 4.3.: Original and preprocessed inputs for each scenario.

Scenario	Original Input	Preprocessed Input
Malware Detection	APK file	Tensor $(1, 30050)$
Cryptocoin Forecasting	Prediction horizon n	Tensor $(n, 8)$
Image Classification	JPEG image	Tensor $(1, 224, 224, 3)$
Sentiment Analysis	Text sequence	Tensor $(n, 1)$

4.6.2. Data Preprocessing

To simulate realistic user interactions, we relied on real-world test data for each scenario. Raw inputs were transformed into structured tensors suitable for the deployed models, as summarized in Table 4.3.

Malware detection. Raw APKs were processed with Drebin to extract binary features, mapped against the 30,050 unique features identified during training. This produced binary vectors of length 30,050, represented as tensors of shape $(1, 30050)$.

Cryptocoin forecasting. Inputs specified the number n of future hourly Bitcoin prices to predict. We applied a MA over the last 24 values of eight correlated altcoins, yielding tensors of shape $(n, 8)$.

Image classification. Joint Photographic Experts Group (JPEG) images were resized to $224 \times 224 \times 3$, resulting in tensors of shape $(1, 224, 224, 3)$[2].

Sentiment analysis. Text sequences were tokenized, lemmatized, and stopwords removed. Tokens were mapped to vocabulary indices, resulting in tensors of shape $(n, 1)$, where n is the sequence length.

Performance evaluation was carried out using oha [199], an HTTP load generator implemented in Rust. Each run lasted one minute with a constant throughput of 200 requests/s. On the server side, 40 parallel threads matched the number of CPU cores.

We tested three payload sizes (small, medium, large) per scenario, yielding twelve experiments per workload: APK size (8.2 KB, 1.4 MB, 45 MB) for malware detection; prediction horizon ($n = 1, 12, 23$) for forecasting; JPEG size (342 KB, 1.4 MB, 18 MB) for image classification; and sequence length (7, 51, 178 tokens) for sentiment analysis. These ranges were selected to reflect realistic lower and upper-bound input sizes observed in practice for each application domain, while allowing us to systematically study the impact of payload size on latency. Results are analyzed below, with a focus on latency distributions.

4.6.3. Micro-Benchmark

The micro-benchmark isolates the inference stage by measuring end-to-end request latencies between client and serving framework. Results are reported for each scenario, with key findings highlighted in boxed annotations (A1-A4) corresponding to research questions Q1-Q4.

[2]The leading dimension corresponds to batch size, set to 1 for single-image inference.

Malware Detection

The minimum average latency across payloads was 0.0372s, 68.37% lower than general-purpose frameworks (0.1176s). The maximum average latency was 0.0895s, 72.04% lower than general-purpose baselines (0.3201s).

> A1/A4: TensorFlow Serving and TorchServe significantly outperformed MLflow, MLServer, and BentoML.

TensorFlow Serving achieved minimum latencies of 0.0215s, 0.0260s, and 0.0242s for small, medium, and large payloads, with maximum latencies (99th percentile) of 0.0431s, 0.0445s, and 0.0432s. TorchServe achieved lower minima (0.0135-0.0136s) but higher maxima (0.1283-0.1434s).

> A3: TorchServe offered lower minimum latencies, but TensorFlow Serving was more stable across percentiles.

> A2: Payload size did not significantly affect latency.

An exception was BentoML, which showed increased 99th-percentile latency for medium-sized APKs, likely due to outliers[3].

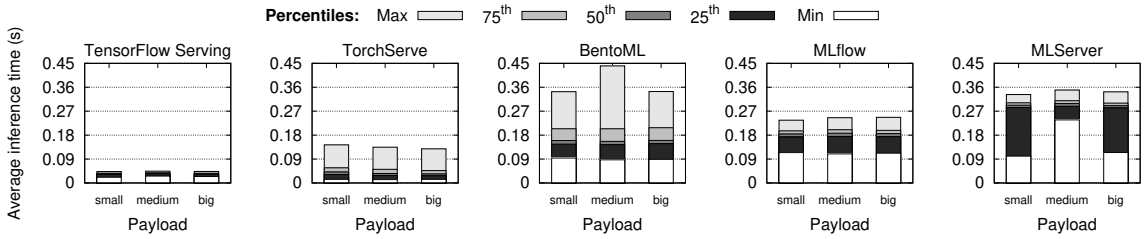

Figure 4.1.: Stacked percentile chart of average inference time for serving frameworks in the malware detection scenario.

Cryptocoin Forecasting

The minimum average latency was 0.0039s, 95.47% lower than general-purpose frameworks (0.0860s). The maximum average latency was 0.0676s, 69.95% lower than baselines (0.2250s).

> A1/A4: TensorFlow Serving and TorchServe clearly outperformed the general-purpose frameworks.

[3]As noted in [200], high-percentile latency variability may arise from shared resources, daemons, queueing, or energy management.

TensorFlow Serving achieved minima of 0.0008-0.0016s and stable maxima (0.0035-0.0042s). TorchServe showed minima of 0.0049-0.0093s but maxima as high as 0.1538s.

> A3: TensorFlow Serving provided both lower latencies overall and greater stability than TorchServe.

> A2: Latencies were not significantly affected by payload size.

Exceptions occurred for BentoML (medium input) and TorchServe (large input), where 99th-percentile latencies spiked due to outliers.

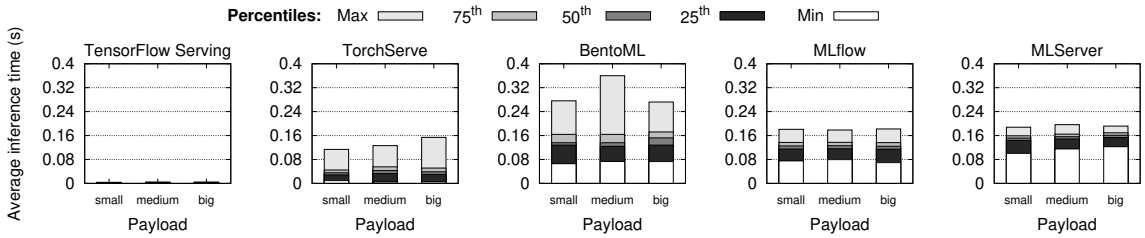

Figure 4.2.: Stacked percentile chart of average inference time for serving frameworks in the cryptocoin forecasting scenario.

Image Classification

The minimum average latency was 0.1427s, 70.2% lower than general-purpose frameworks (0.4788s). The maximum average latency was 0.4828s, 80.06% lower than baselines (2.4213s).

> A1/A4: TensorFlow Serving and TorchServe again outperformed MLflow, MLServer, and BentoML.

TensorFlow Serving achieved minima of 0.0706-0.0756s with maxima (99th percentile) of 0.4250-0.4444s. TorchServe showed higher minima (0.2065-0.2188s) and maxima (0.5272-0.5384s).

> A3: TensorFlow Serving exhibited lower latencies overall, though both frameworks displayed instability across percentiles.

> A2: Latencies were largely unaffected by payload size.

BentoML and MLflow displayed increased 99th-percentile latency for small (BentoML) and medium inputs (both frameworks), again likely due to outliers.

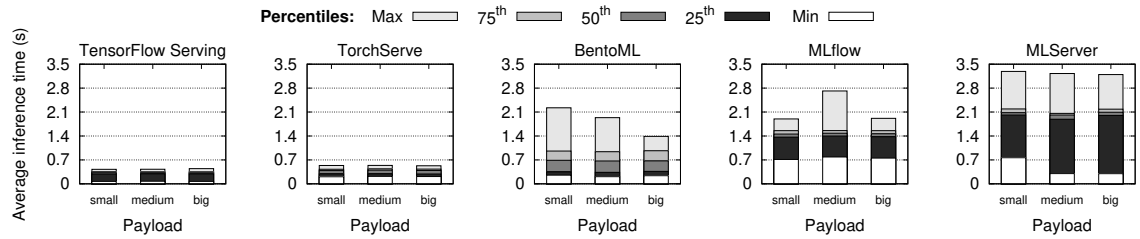

Figure 4.3.: Stacked percentile chart of average inference time for serving frameworks in the image classification scenario.

Sentiment Analysis

The minimum average latency was 0.0042s, 95.44% lower than general-purpose frameworks (0.0921s). The maximum average latency was 0.1231s, 53.19% lower than baselines (0.2630s).

> A1/A4: TensorFlow Serving and TorchServe again outperformed the general-purpose frameworks.

TensorFlow Serving achieved minima of 0.0013-0.0023s with maxima (99th percentile) of 0.0031-0.0046s. TorchServe showed minima of 0.0032-0.0087s but maxima as high as 0.3177s.

> A3: TensorFlow Serving outperformed TorchServe in both latency and stability.

> A2: Latencies were not significantly influenced by payload size.

Exceptions were observed for BentoML and TorchServe, which showed higher 99th-percentile latencies for small and medium inputs compared to large ones.

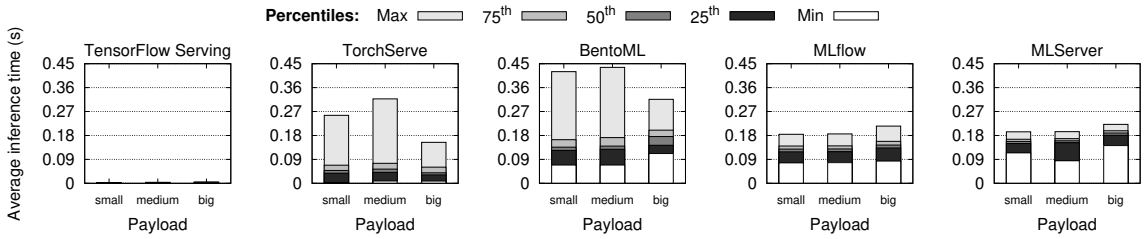

Figure 4.4.: Stacked percentile chart of average inference time for serving frameworks in the sentiment analysis scenario.

4.6.4. Macro-Benchmark

While the micro-benchmark isolates framework inference latencies, real-world deployments include preprocessing and postprocessing stages. To capture this broader perspective, we conducted a macro-benchmark simulating end-to-end pipelines.

We implemented a lightweight local server in Flask [177] performing: *(i)* request reception, *(ii)* input preprocessing, *(iii)* forwarding to the serving framework via its API, and *(iv)* result postprocessing and response.

This setup reflects a common deployment pattern in practice, where application logic and model inference are decoupled and connected via HTTP-based interfaces. While production systems may employ more complex stacks (*e.g.*, asynchronous processing, message queues, or service meshes), the chosen design captures the dominant sources of end-to-end latency and allows for a controlled and reproducible comparison across frameworks.

Results are summarized below.

Malware Detection

As shown in Figure 4.5, end-to-end latencies were dominated by feature extraction, which introduced high overhead and cold starts. Framework differences were marginal, though TensorFlow Serving and TorchServe achieved slightly shorter turn-around times. Large APKs produced markedly higher latencies due to heavier preprocessing.

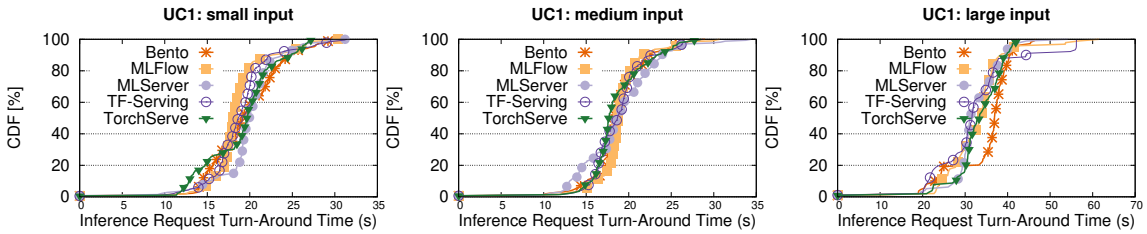

Figure 4.5.: Cumulative distribution function of request turn-around time in the malware detection scenario.

Cryptocoin Forecasting

In the forecasting scenario (Figure 4.6), turn-around times were consistently short across all frameworks. Differences were negligible, with TensorFlow Serving and TorchServe slightly ahead. Payload size (number of steps) had no meaningful effect, reflecting the lightweight preprocessing.

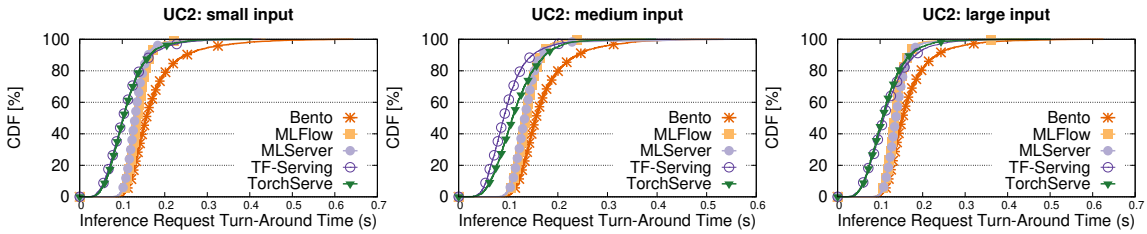

Figure 4.6.: Cumulative distribution function of request turn-around time in the cryptocoin forecasting scenario.

Image Classification

As shown in Figure 4.7, image preprocessing introduced significant overhead, visible as cold starts across all frameworks. While most frameworks performed similarly, MLServer exhibited slightly higher turn-around times. Input size played a clear role: larger JPEGs led to markedly higher latencies due to heavier preprocessing pipelines.

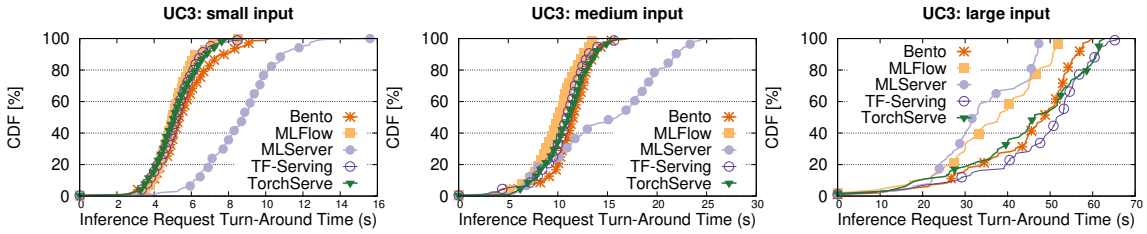

Figure 4.7.: Cumulative distribution function of request turn-around time in the image classification scenario.

4.7. Lessons Learned

Our evaluation highlights clear differences in efficiency across serving frameworks. Across all scenarios, DL-specific frameworks (TensorFlow Serving and TorchServe) consistently outperformed the three general-purpose solutions (MLflow, MLServer, BentoML). This finding directly answers Q1 and Q4, showing that specialization to a particular DL ecosystem remains a decisive advantage in practice.

Among the frameworks, TensorFlow Serving achieved the best overall results, delivering the lowest inference latencies in every workload. A key factor is that all models in our study were originally implemented in TensorFlow, enabling TensorFlow Serving to leverage native optimizations and seamless compatibility with the `SavedModel` format. This represents a limitation of the present evaluation, as different performance rankings might be observed if models were instead natively implemented in PyTorch. However, selecting a single training ecosystem and

relying on ONNX-based conversion was necessary to ensure a controlled and systematic comparison across frameworks at this stage.

TorchServe ranked second, benefiting from native support for PyTorch models. Since our evaluation required converting TensorFlow models into PyTorch format via ONNX, TorchServe did not operate on its ideal artifacts. Nevertheless, the converted models preserved all weights and parameters, yielding identical predictions and competitive performance.

When comparing the two DL-specific frameworks, TensorFlow Serving demonstrated greater stability across latency percentiles, whereas TorchServe exhibited higher variability and occasional outliers (see Figures 4.1, 4.2, and 4.4). This distinction answers Q2 and Q3: while both frameworks achieved low latencies, TensorFlow Serving provided the most reliable performance under varying conditions.

In summary, our results suggest that practitioners should favor TensorFlow Serving when deploying TensorFlow models and TorchServe when working with PyTorch. Despite the growing flexibility of general-purpose frameworks, specialized solutions remain the most efficient open-source options for serving pretrained DL models in production.

4.8. Summary and Next Steps

This chapter provided a systematic evaluation of five state-of-the-art model-serving frameworks (TensorFlow Serving, TorchServe, MLServer, MLflow, and BentoML) across four representative application scenarios: malware detection, cryptocoin forecasting, image classification, and sentiment analysis. The study was guided by four research questions centered on inference latency, the most critical factor for the usability of deployed ML services.

Our results offered consistent answers: DL-specific frameworks (TensorFlow Serving and TorchServe) clearly outperformed general-purpose solutions (MLflow, MLServer, BentoML). Among them, TensorFlow Serving achieved the lowest latencies and the most stable performance, owing to its tight integration with the TensorFlow ecosystem and efficient handling of `SavedModel` artifacts. TorchServe followed as the second-best option, though with greater variability. Together, these findings underscore the advantages of specialized serving solutions for latency-sensitive workloads, thereby addressing Q1-Q4.

Finally, we leave the investigation of end-to-end serving platforms based on Kubernetes [201], including systems such as KServe [202] and Kubeflow [203], to future extensions of this work.

Part III.

Machine Learning for Decentralized Finance

Chapter 5.

Cryptocoin Price Forecasting Using Correlation Patterns

In this chapter, we examine correlation and causality patterns in cryptocurrency markets and assess their value for forecasting. By analyzing cross-asset dependencies among 62 cryptocoins, with particular focus on Bitcoin and Ether, we identify altcoins that act as reliable predictors of main-coin price movements. Building on these insights, we evaluate Machine Learning models for time-series forecasting, showing that Gradient-Boosting ensembles consistently outperform Recurrent Neural Networks. The findings provide both a characterization of interdependencies across digital assets and practical guidance for predictive modeling in volatile markets.

Chapter Outline

5.1. Introduction

Cryptocurrencies, or cryptocoins, are digital assets secured through cryptographic techniques that ensure the integrity and transparency of transactions on distributed ledgers. Since the launch of Bitcoin (BTC) in 2009 [17], thousands of cryptocoins have emerged, including Ether (ETH), the native currency of Ethereum [18], and numerous altcoins [204] such as Litecoin (LTC) [120], Cardano (ADA) [121], and Ripple (XRP) [122]. These assets are traded on centralized and decentralized exchanges such as Coinbase [127], Binance [129], and Uniswap [130], supporting a global economy whose market capitalization has reached levels comparable to national economies [126].

Despite their growth, cryptocoin markets remain highly volatile. Prices can fluctuate dramatically within short intervals, driven by speculative activity, limited regulation, and the influence of large stakeholders ("whales"). Historical data reveal pronounced volatility and frequent comovements across assets, as discussed in Section 3.2.2 of Chapter 3 (Figure 3.3).

Understanding these correlations is essential for both research and practice. Market platforms increasingly expose correlation indicators to end-users, often reporting simplified metrics such as the Pearson coefficient r [205] over recent windows [132]. While widely used to guide trading decisions, such indicators provide a limited perspective and are typically disclosed without methodological detail. A more systematic study is needed to assess their reliability and to clarify their role in forecasting.

In this chapter, we investigate correlation and causality relationships among a broad set of cryptocoins, focusing on BTC and ETH, the two leading assets by market capitalization. We first quantify the extent to which price trends of major coins align with those of altcoins. Building on this analysis, we explore whether correlation patterns can be exploited for forecasting: specifically, whether altcoin trajectories improve the prediction of BTC and ETH price movements. To this end, we evaluate a range of time-series forecasting methods, including Gradient-Boosting Machines (GBMs) and Recurrent Neural Networks (RNNs), and compare their performance against baseline predictors.

The contributions of this study [52, 53] are threefold:

- An empirical characterization of correlation patterns across 62 cryptocoins over multiple time scales;
- A causality analysis identifying directional dependencies between altcoins and major coins;
- The design and evaluation of forecasting models that exploit these relationships, demonstrating that Machine Learning (ML)-based approaches significantly outperform naive baselines.

All datasets and code are publicly released to ensure transparency and reproducibility [58]. By examining how correlation structures shape the dynamics of the cryptocoin market, this chapter advances the understanding of interdependencies across digital assets and demonstrates their utility for practical forecasting.

5.2. Related Work

The analysis of co-movements and cross-correlations in cryptocurrency markets has attracted increasing attention in recent years. Early studies primarily examined volatility dynamics of major assets. Katsiampa [23], for instance, investigated BTC and ETH, reporting interdependencies between the two and sensitivity to market news. Aslanidis et al. [24] observed broadly similar correlation patterns across cryptocoins, though with unstable trends over time. Their study also compared cryptocoins such as BTC, XRP, Dash (DASH), and Monero (XMR) with traditional assets including the S&P 500, U.S. Treasury bonds, and gold, finding largely independent behavior relative to conventional financial markets. Kumar et al. [206] identified BTC as the market leader using wavelet-based methods, showing that its price movements quickly propagate to other coins. Chaudhari et al. [207] further analyzed global market behavior and uncovered distinct, though non-persistent, community structures characterized by cross-correlation.

Parallel efforts compared forecasting approaches spanning traditional statistical models and modern ML. Siami-Namini et al. [208] benchmarked the Autoregressive Integrated Moving Average (ARIMA) model [209] against RNNs, confirming the superior predictive power of Long Short-Term Memory (LSTM) architectures. AutoML has also been explored: Shah et al. [210] introduced a system that automatically selects and optimizes forecasting models, reducing the need for manual configuration.

Beyond purely price-based predictors, external signals have also been leveraged. Kraaijeveld and De Smedt [211], for example, incorporated Twitter activity to forecast price dynamics, demonstrating predictive power for BTC, LTC, and Bitcoin Cash (BCH).

In summary, prior research has established the existence of correlations and causal linkages between major cryptocoins and shown the effectiveness of advanced ML methods for time-series forecasting. However, existing studies typically address correlation analysis or forecasting in isolation and focus on a limited set of assets. In contrast, our work provides a systematic characterization of correlations across 62 cryptocoins and investigates how such relationships can be exploited to improve forecasting accuracy for leading assets.

5.3. Datasets

This study pursues two main objectives: *(i)* to characterize correlation patterns across a broad set of cryptocoins, and *(ii)* to investigate causality relationships and forecast the price dynamics of the two leading assets, BTC and ETH, using highly correlated altcoins ($r \geq 0.6$). The first objective requires wide coverage to capture global market trends, while the second calls for high-resolution time series that enable robust forecasting. To meet these requirements, we rely on two complementary datasets spanning 33 months (20 February 2020 to 5 November 2022). For forecasting, we additionally reserve a holdout period (6 November 2022 to 26 February 2023), used exclusively for evaluation to prevent data leakage.

CoinMarketCap dataset. The first dataset was collected from CoinMarketCap [204], a leading aggregator of cryptocurrency market data. From over 9,000 listed coins, we selected the

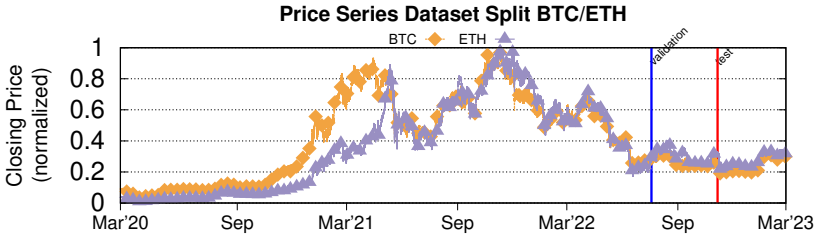

Figure 5.1.: Closing prices of BTC and ETH with validation/test split.

top 62 by trading volume, excluding stablecoins since their peg to reserve assets prevents them from reflecting genuine market dynamics. For each coin, we obtained daily records of standard financial variables: high and low prices (intra-day maximum and minimum), open and close prices, and traded volume. The dataset comprises 61,380 observations, corresponding to 990 daily steps per asset. As expected, BTC and ETH dominate trading volume and serve as benchmarks throughout the study. Their closing price series exhibit strong correlation, with $r \approx 0.9$ over the observation window (Figure 5.1).

Binance dataset. The second dataset, obtained from Binance [129], provides high-resolution intraday data at one-minute granularity. It covers the 16 most actively traded cryptocoins: Cardano (ADA), Basic Attention Token (BAT), Binance Coin (BNB), Bitcoin (BTC), Dash (DASH), Dogecoin (DOGE), Ether (ETH), Chainlink (LINK), Litecoin (LTC), Neo (NEO), Defiance Quantum ETF (QTUM), Tronix (TRX), Stellar Lumens (XLM), Monero (XMR), Ripple (XRP), and Zcash (ZEC). The dataset contains 25,218,800 individual records, with each coin represented by a time series of 1,576,175 steps. This fine-grained resolution is essential for evaluating forecasting models under realistic market conditions.

5.4. Correlation Analysis

We begin by analyzing cross-correlation patterns in the CoinMarketCap dataset to assess the degree of alignment between the two main coins (BTC and ETH) and the wider set of altcoins. For each asset, we consider five variables (High, Low, Open, Close, aggregated into an average OHLC, and trading Volume) and compute the Pearson correlation coefficient r against BTC and ETH. Results are summarized in Figure 5.2, which presents correlograms across 60 altcoins.

To capture temporal dynamics, we compute correlations over daily, weekly, and monthly horizons. For weekly and monthly resolutions, both sliding and tumbling windows are applied: sliding windows allow partial overlap across observations and provide smoother, more temporally responsive estimates, while tumbling windows rely on disjoint partitions and yield independent, non-overlapping correlation measurements. Across all cases, correlations between major coins and altcoins remain consistently strong, with most coefficients close to 1. A notable

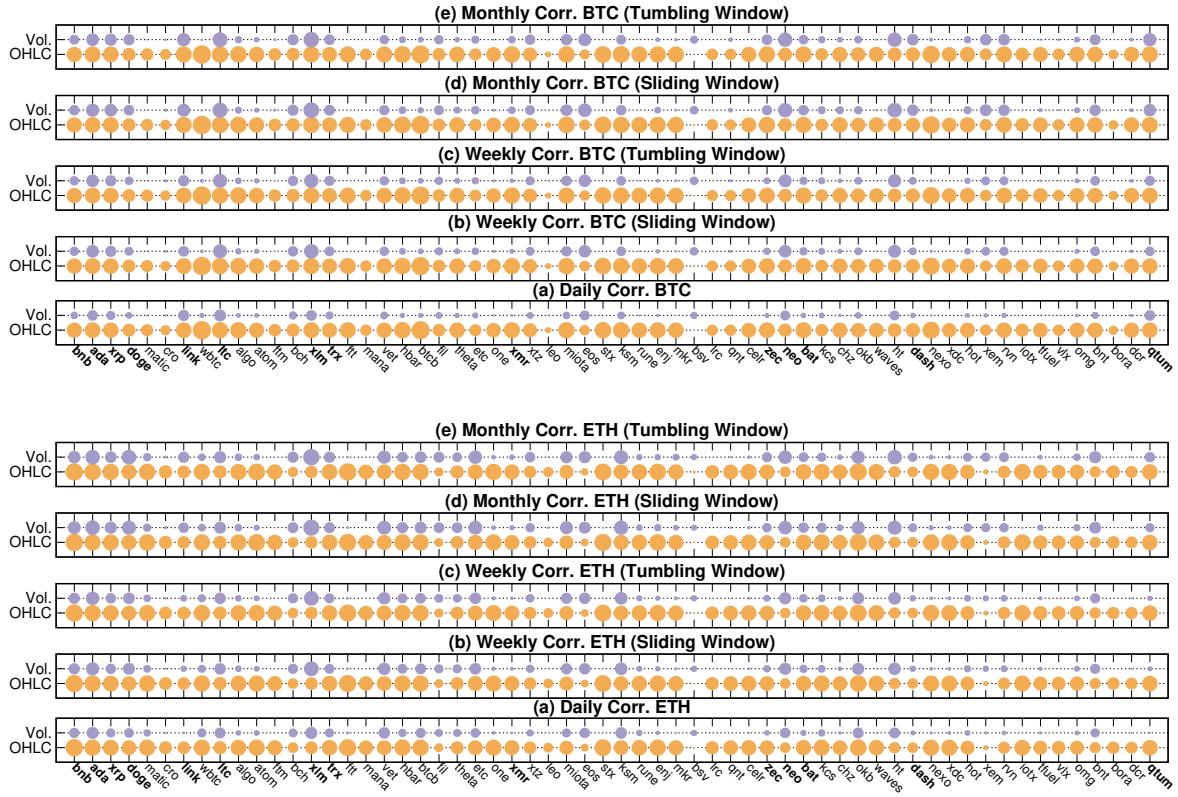

Figure 5.2.: Correlogram between the average OHLC price and volume of 60 altcoins and BTC (top) and ETH (bottom). Coins in bold are used in the causality analysis.

exception is Bitcoin Satoshi Vision (BSV), which exhibits markedly weaker alignment, reflecting its divergent market behavior.

Sliding and tumbling windows produce broadly similar outcomes, with stronger correlations generally observed at weekly or monthly resolution. Correlograms for BTC and ETH are also closely aligned, suggesting that both main coins exert comparable influence over altcoin dynamics.

To illustrate individual trajectories, Figure 5.3 shows daily correlation trends between BTC or ETH and four representative altcoins. Each case exemplifies a distinct level of correlation intensity: weak (0-0.25), average (0.25-0.5), above average (0.5-0.75), and high (0.75-1). The Pearson coefficient $r(A,B)_{t_n}$ between a main coin A and an altcoin B is computed at each time step t_n based on the previous $n-1$ observations, with $r(A,B)_{t_0}$ set to 1 by convention.

For BTC, correlations with NEO and LTC remain stable and consistently close to 1, whereas BSV and BORA Token (BORA) display more irregular trends, with seasonal fluctuations in late 2020 and early 2021. From mid-2021 onward, BORA increasingly aligns with the stronger group, while BSV diverges further. ETH shows similar stability with EOS (EOS), LINK, and BNB, although EOS weakens temporarily between late 2020 and early 2021, and again in late

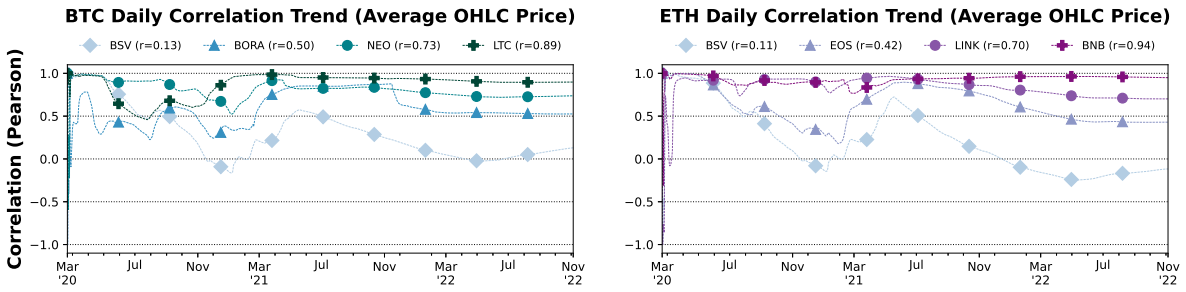

Figure 5.3.: Daily cross-correlation trends between BTC (left) and ETH (right) and four representative altcoins, showing different correlation intensity classes.

2021. ETH-BSV correlations also reveal seasonal patterns, alternating between decline and recovery phases.

These behaviors are not random, but reflect structural differences across assets, including variations in market capitalization, liquidity, investor base, and coupling to broader market trends. Highly correlated coins typically share similar trading dynamics and respond coherently to global market shocks, whereas weaker or unstable correlations often correspond to assets with idiosyncratic demand, lower liquidity, or protocol-specific events that decouple them from market leaders.

Based on these findings, we select 14 altcoins that exhibit strong correlations ($r \geq 0.6$) with BTC and ETH for further causality analysis in Section 5.5. For these assets, the Binance dataset provides higher temporal resolution, enabling a more detailed investigation of directional relationships and forecasting potential.

5.5. Causality Analysis

Strong correlation patterns between altcoins and main coins are not sufficient to establish cause-and-effect relationships. To address this, we investigate Granger causality [212] between 14 strongly correlated altcoins (ADA, BAT, BNB, DASH, DOGE, LINK, LTC, NEO, QTUM, TRX, XLM, XMR, XRP, ZEC) and the two market leaders, BTC and ETH. For this analysis, we use a modified version of the Binance dataset, where per-minute observations are aggregated into daily windows to ensure consistency across assets.

We estimate one-directional Granger causality using the Toda-Yamamoto (T-Y) approach [213]. For a given pair (A_c, M_c), where A_c is an altcoin and $M_c \in \{\text{BTC, ETH}\}$, we test whether A_c can be used as a predictor of M_c. The null hypothesis is defined as:

$$H_0 : A_c \nRightarrow M_c$$

Given d, the maximum order of integration across the two series, we construct a Vector Autoregression (VAR) model [214], VAR$(p+d)$, and test whether the coefficients of the first p lags

Table 5.1.: Results of the m-Wald test for Granger causality in Bitcoin (left) and Ether (right) price series. Legend: H_0 rejected (✗) or not rejected (✓). H_0 is rejected if $p \leq 0.05$.

H_0	χ^2	p-value	Result
ADA ⇏ BTC	31.31	0.018	✗
BAT ⇏ BTC	83.98	5.8e-5	✗
BNB ⇏ BTC	5.29	0.15	✓
DASH ⇏ BTC	19.06	0.014	✗
DOGE ⇏ BTC	69.43	0.001	✗
LINK ⇏ BTC	8.85	0.012	✗
LTC ⇏ BTC	120.43	9.2e-11	✗
NEO ⇏ BTC	55.82	5.4e-5	✗
QTUM ⇏ BTC	59.48	2.6e-5	✗
TRX ⇏ BTC	84.77	5.2e-6	✗
XLM ⇏ BTC	69.43	0.001	✗
XMR ⇏ BTC	46.69	0.027	✗
XRP ⇏ BTC	77.05	3.9e-4	✗
ZEC ⇏ BTC	47.14	3.7e-7	✗

H_0	χ^2	p-value	Result
ADA ⇏ ETH	89.75	2.2e-8	✗
BAT ⇏ ETH	91.27	4.2e-8	✗
BNB ⇏ ETH	18.76	0.04	✗
DASH ⇏ ETH	70.56	1.4e-7	✗
DOGE ⇏ ETH	134.51	2.7e-13	✗
LINK ⇏ ETH	23.53	0.002	✗
LTC ⇏ ETH	65.20	1.1e-6	✗
NEO ⇏ ETH	142.76	1.8e-13	✗
QTUM ⇏ ETH	117.14	5.7e-15	✗
TRX ⇏ ETH	41.60	0.001	✗
XLM ⇏ ETH	134.51	2.7e-13	✗
XMR ⇏ ETH	73.65	1.6e-5	✗
XRP ⇏ ETH	145.04	7.9e-14	✗
ZEC ⇏ ETH	140.84	1.2e-14	✗

of A_c are zero in the M_c equation. This corresponds to applying a modified Wald (m-Wald) test [215], which follows a χ^2 distribution with p degrees of freedom.

To determine d, we performed stationarity checks using the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) [216] and Augmented Dickey-Fuller (ADF) [217] tests. In all cases, $d = 1$, indicating that first-order differencing was sufficient for stationarity. The optimal lag length p was selected for each pair based on the Akaike Information Criterion (AIC) [218].

Table 5.1 summarizes the results. With the exception of BNB, all considered altcoins Granger-cause BTC. For ETH, all 14 altcoins exhibit Granger causality.

These findings motivate our forecasting experiments in Section 5.6, where the identified Granger-causing altcoins are used as feature variables for predictive models of main-coin closing prices.

5.6. Evaluation

This section presents our experimental evaluation. We forecast BTC and ETH closing prices using the price series of correlated altcoins identified in Section 5.5. The predictor sets include {ADA, BAT, DASH, DOGE, LINK, LTC, NEO, QTUM, TRX, XLM, XMR, XRP, ZEC} for BTC, and {ADA, BAT, BNB, DASH, DOGE, LINK, LTC, NEO, QTUM, TRX, XLM, XMR, XRP, ZEC} for ETH. We then evaluate alternative ML-based forecasting methods to determine the most effective approach for predicting main-coin price trends.

Table 5.2.: Optimized hyperparameters for GBM and RNN models in BTC and ETH forecasting.

Model	BTC lr	ETH lr	BTC iters	ETH iters
XGBoost	0.01	0.01	52	384
LightGBM	0.01	0.10	39	52
CatBoost	0.10	0.10	5	24
LSTM	0.001	0.001	3	3
GRU	0.001	0.001	4	32

5.6.1. Experimental Setup

Experiments were conducted on a workstation running Ubuntu 22.04.2 with Linux kernel 5.19.0-35-generic, equipped with an 8-core AMD Ryzen® 7 5700G CPU at 3.8 GHz, 32 GB RAM, and an NVIDIA GeForce RTX® 3070 GPU. RNNs, including LSTM and Gated Recurrent Unit (GRU), were implemented in PyTorch 1.11.0 (CUDA 11.3) with PyTorch Lightning 1.5.10. GBMs were implemented using XGBoost 1.5.1, LightGBM 3.3.2, and CatBoost 1.0.5.

5.6.2. Model Training and Validation

The Binance dataset was split into three partitions (see Figure 5.1): a training set of 1,260,941 observations (20 Feb 2020-19 Jul 2022), a validation set of 157,617 observations (20 Jul 2022-5 Nov 2022), and a final test set of 157,617 observations (6 Nov 2022-26 Feb 2023), reserved exclusively for evaluation.

For GBMs, model selection focused on learning rate and the optimal number of iterations (trees), with optimization based on Mean Squared Error (MSE). Learning rates were chosen via grid search across $\{10^{-i} \mid i = 1, \ldots, 6\}$, with 10-fold cross-validation. The number of trees was capped at 500, with early stopping after 20 rounds without validation improvement.

RNNs were trained with MSE loss and the Adam optimizer [219]. Both LSTM and GRU models used $n = 10$ hidden layers with $m = 100$ neurons per layer, batch size of 256, and learning rates tuned via a warm-up schedule starting at 10^{-6}. Training was capped at 500 epochs with early stopping after 20 stagnant rounds.

Validation losses converged after an average of 32 iterations for BTC and 153 for ETH in GBMs, defining the checkpoints for optimal models (Figure 5.4). For RNNs, convergence was reached within only a few epochs, reflecting the moderate complexity of the underlying dynamics.

5.6.3. Price Series Forecasting

We now compare forecasts against ground-truth price series in the test period. Figure 5.5 illustrates predicted closing prices for BTC and ETH.

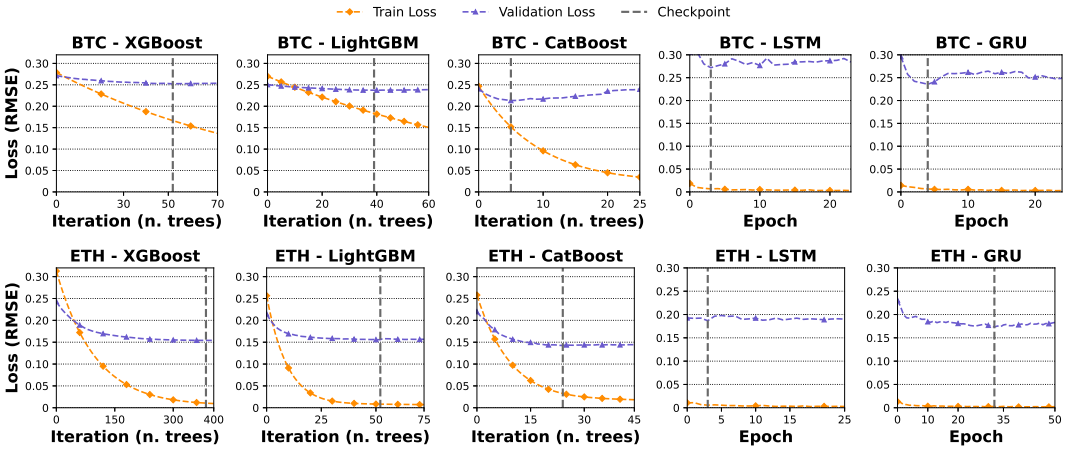

Figure 5.4.: Training and validation loss trends for BTC (top) and ETH (bottom) closing price forecasting.

Table 5.3.: Comparison of MSE, RMSE, MAE and MAPE for BTC and ETH forecasting (best models in bold).

Model	MSE_{BTC}	MSE_{ETH}	$RMSE_{BTC}$	$RMSE_{ETH}$	MAE_{BTC}	MAE_{ETH}	$MAPE_{BTC}$	$MAPE_{ETH}$
XGBoost	10,035,762	**13,891**	3,167	**118**	2,859	**102**	0.16	**0.07**
LightGBM	**8,865,905**	14,321	**2,977**	120	**2,768**	102	**0.15**	0.07
CatBoost	9,036,512	15,127	3,006	123	2,746	104	0.15	0.07
LSTM	14,604,921	20,613	3,821	144	3,319	117	0.18	0.08
GRU	16,415,682	18,167	4,051	135	3,735	106	0.21	0.08
Mean Regr.	137,296,033	197,503	11,717	444	11,330	401	0.63	0.31
Median Regr.	138,962,435	122,789	11,788	350	11,403	295	0.63	0.24

All models provide reliable forecasts, consistently outperforming naive mean and median baselines. The GBMs (XGBoost, LightGBM, CatBoost) capture both stable regimes and volatile transitions, with only minor drifts near sharp shifts. RNNs tend to oversmooth stable periods, as seen in BTC forecasts between Dec 2022 and Feb 2023.

5.6.4. Analysis of Results

Performance was quantified using MSE, Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE), alongside two deterministic baselines (mean and median regressors). Results are shown in Table 5.3.

All models substantially outperform the deterministic baselines. GBMs achieve the strongest results: for BTC, their average RMSE is 3,050 (22% lower than RNNs and 74% lower than baselines); for ETH, their average RMSE is 120 (14% lower than RNNs and 70% lower than baselines). Among GBMs, LightGBM performs best for BTC, while XGBoost is best for ETH.

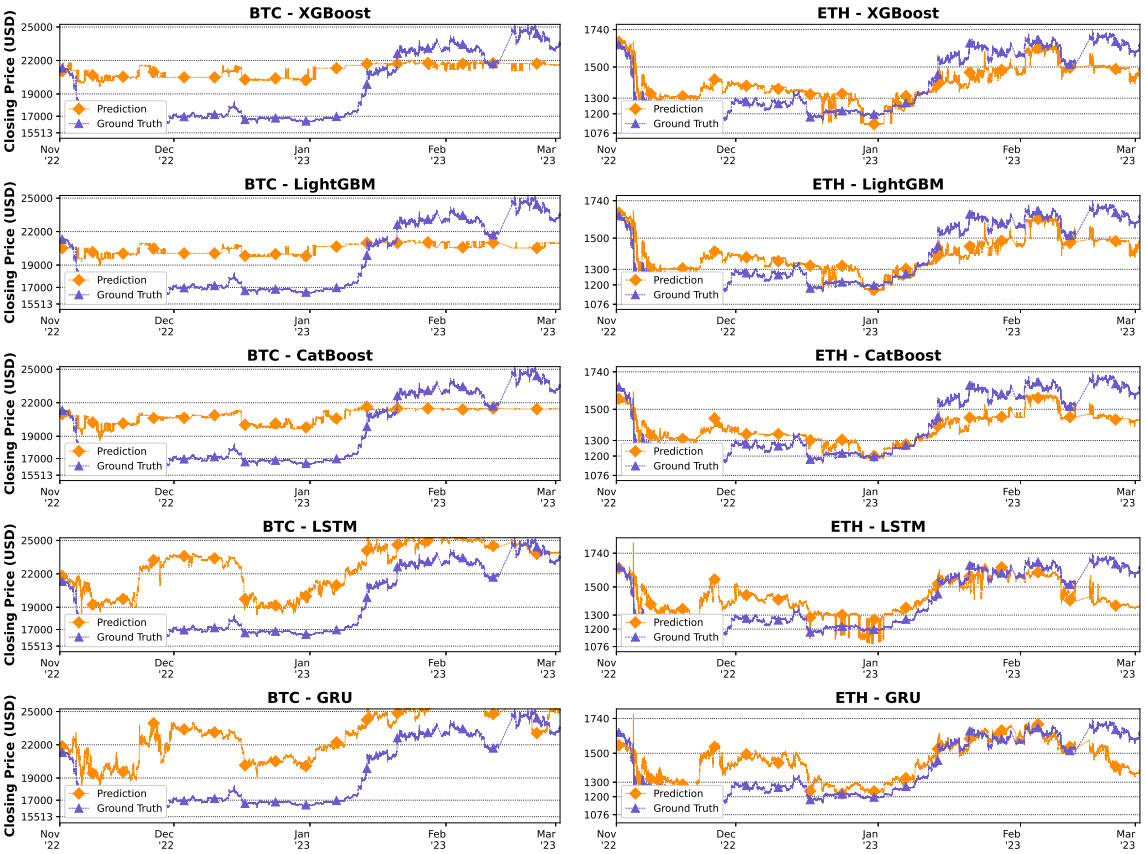

Figure 5.5.: Forecasted closing prices for BTC (left) and ETH (right) in the test period.

While these results do not imply precise or error-free prediction in absolute terms, they demonstrate that correlation-informed ML models consistently extract more predictive signal than naive statistical baselines, yielding measurable and reproducible improvements across all error metrics.

> Correlation-driven feature selection combined with GBM-based predictors provides consistent improvements over naive baselines and recurrent models, making it a practical and effective strategy for cryptocurrency price forecasting.

5.7. Lessons Learned

Our study yields two main insights.

First, cryptocoin markets, despite their pronounced volatility, display clear and persistent correlation structures linking major assets such as BTC and ETH to a broad set of altcoins. Correlation alone, however, is insufficient: by applying the T-Y approach to Granger causality, we identified a subset of altcoins that act as reliable predictors of main-coin price dynamics.

Second, the evaluation highlights a methodological lesson. Across all experiments, GBMs (*e.g.*, XGBoost, LightGBM, CatBoost) consistently outperformed recurrent architectures (LSTM, GRU), delivering lower prediction errors with reduced training complexity.

It is important to note that the deterministic baselines (mean and median regressors) are intentionally simple and serve as lower-bound references rather than competitive forecasting models. Their role is to contextualize performance gains and verify that the learned models capture nontrivial temporal structure beyond static predictors.

5.8. Summary and Next Steps

This chapter presented a systematic study of correlation and causality in cryptocurrency markets, covering 62 assets over nearly three years of activity. Our contributions are threefold:

- An empirical characterization of cross-asset correlations, revealing both persistent alignments and notable exceptions;
- A causality analysis showing that several altcoins Granger-cause BTC and ETH price movements;
- A comparative evaluation of forecasting models, demonstrating that GBMs consistently outperform recurrent architectures in accuracy and efficiency.

Looking forward, several avenues of research emerge.

First, future work will extend the forecasting setup by incorporating autoregressive components of the target series itself, combining past values of BTC or ETH with correlated altcoin features to assess whether joint autoregressive-cross-asset models yield further performance gains.

Second, given that forecasts for ETH exhibited lower error levels and greater stability across models, future studies should further investigate the underlying causes of this behavior, disentangling the roles of price scale, volatility, market structure, and data characteristics.

Third, while this study focused on causality from altcoins to main coins for forecasting purposes, an explicit analysis of the reverse direction (namely, whether BTC and ETH Granger-cause altcoin price dynamics) represents a natural extension to better characterize leader-follower relationships in cryptocurrency markets.

Fourth, we plan to extend the analysis beyond cryptocurrency markets, incorporating traditional financial indices and commodities to capture cross-market dependencies. Preliminary evidence, such as the correlation between BTC and the NASDAQ index, motivates this direction.

Fifth, we will investigate other domains where correlation and causality play predictive roles, including e-health (*e.g.*, early anomaly detection from body sensor data) and environmental monitoring (*e.g.*, earthquake forecasting).

Finally, we remain committed to open science: all datasets and code will continue to be released publicly [58], supporting reproducibility and enabling further advances in forecasting for volatile and interdependent markets.

Chapter 6.

CryptoAnalytics: a Toolkit for Cryptocoin Price Forecasting

This work introduces CryptoAnalytics, a toolkit that operationalizes the insights of Chapter 5. It consolidates correlation-driven forecasting methods and state-of-the-art ML models into a unified software system, offering both reproducibility for researchers and practical utility for end users. Through its modular design and integration with modern serving frameworks, CryptoAnalytics bridges the gap between theoretical analysis and real-world deployment of cryptocoin prediction services.

Chapter Outline

6.1. Introduction

Cryptocoin markets are marked by rapid fluctuations, high volatility, and complex interdependencies among assets. While these dynamics make forecasting challenging, recent research has shown that price trajectories of different coins frequently exhibit co-movement and cross-correlation [23, 24]. These findings motivate the development of dedicated tools that can systematically exploit such relationships to produce reliable forecasts.

To this end, we introduce CryptoAnalytics [54], a toolkit designed to support cryptocoin price forecasting through correlation-driven modeling and state-of-the-art ML techniques. CryptoAnalytics integrates GBMs, including XGBoost, LightGBM, and CatBoost, and RNNs, including LSTM and GRU, within a unified and extensible framework. By doing so, it enables diverse stakeholders, ranging from individual investors to institutions and regulators, to leverage correlated assets as predictors of major-coin price dynamics.

The design of CryptoAnalytics is guided by three goals:

- **Accessibility:** a straightforward interface lowers the barrier to entry for non-expert users;
- **Flexibility:** multiple learning algorithms and customizable predictor sets accommodate a variety of forecasting needs;
- **Deployability:** the system is engineered for integration into production environments, serving as the basis for real-time forecasting services.

CryptoAnalytics has already been employed in peer-reviewed studies [52, 53], whose results were presented in Chapter 5. In that context, it proved effective in uncovering correlation patterns across dozens of cryptocoins and in forecasting BTC and ETH prices with high accuracy. Whereas Chapter 5 developed the theoretical and empirical foundations of correlation-driven forecasting, CryptoAnalytics embodies their concrete implementation, consolidating those insights into a practical and extensible software system.

By integrating the methods introduced earlier into a unified toolkit, CryptoAnalytics contributes both to reproducible research and to the creation of robust, ready-to-use services for cryptocoin price prediction.

6.2. Related Work

The growing interest in cryptocoins as investment assets has led to the emergence of numerous forecasting services. Popular platforms such as WalletInvestor [220], CryptoPredictions [221], and DigitalCoinPrice [222] provide daily, monthly, and yearly forecasts for hundreds or even thousands of coins. These systems typically combine market indicators such as exchange rates, trade volumes, and historical volatilities, often through regression-based or proprietary ML models. However, their internal pipelines are rarely documented in detail, reproducibility is limited, and full functionality is frequently locked behind paid subscriptions.

In contrast, CryptoAnalytics is an open-source toolkit that consolidates state-of-the-art ML models for reproducible cryptocoin forecasting. Its design differs from existing services in three key respects. First, its transparency and accessibility make it suitable for both research and practice, while also supporting external contributions such as new data connectors or additional algorithms. Second, rather than relying solely on autoregressive features of the target coin, CryptoAnalytics leverages correlation-driven predictors, *i.e.*, price series of strongly correlated altcoins, a strategy shown in Chapter 5 to significantly improve forecasts for BTC and ETH. Third, it integrates a diverse set of forecasting approaches, ranging from GBMs (XGBoost, LightGBM, CatBoost) to RNNs (LSTM, GRU), enabling systematic benchmarking within a unified workflow.

6.3. The CryptoAnalytics Toolkit

6.3.1. System Requirements

CryptoAnalytics runs on standard desktop and server environments without strict hardware requirements. Since training RNNs can be computationally intensive, the toolkit automatically detects CUDA-capable [223] Graphics Processing Units (GPUs) and leverages them when available. If no GPU is present, the entire pipeline executes on the CPU, ensuring portability. The toolkit requires Python v3.9.10.

6.3.2. Software Architecture

CryptoAnalytics implements a cryptocoin forecasting pipeline exposed through a Command-Line Interface (CLI). The workflow is organized into five steps: *(i)* data pull, *(ii)* data split, *(iii)* model pretrain, *(iv)* model forecast, and optionally *(v)* correlation analysis. The overall execution flow is shown in Fig. 6.1, where each step is represented as a modular component that can be executed independently or combined into an end-to-end pipeline. This modular architecture facilitates reproducibility and extensibility, making it straightforward to integrate new data sources, models, or evaluation methods.

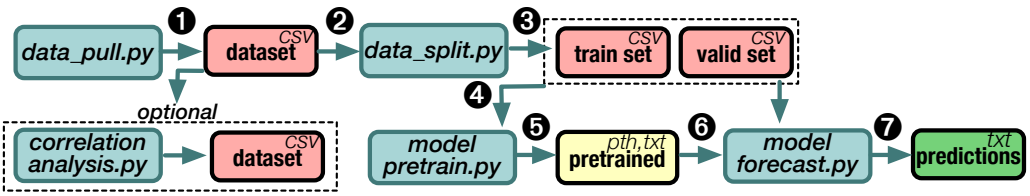

Figure 6.1.: Workflow and core functionalities of CryptoAnalytics.

6.3.3. Core Functionalities

The functionalities of CryptoAnalytics map directly onto the forecasting pipeline shown in Fig. 6.1. Each step is implemented as a CLI command with configurable arguments.

Data pull (Fig. 6.1-❶) generates datasets of Open-High-Low-Close (OHLC) cryptocoin prices from CoinMarketCap [204]. OHLC charts are widely used in financial markets to represent price movements over time. Configurable arguments include the destination directory, output file name, list of coins (in `.json` format), and start/end dates (in %d-%m-%Y format). The resulting dataset is passed as input (Fig. 6.1-❷) to the next step.

Data split (Fig. 6.1-❸) partitions the dataset into training and validation subsets. Both average OHLC and Close price can be used as the target variable. The training set is used during model fitting (Fig. 6.1-❹), while the validation set supports hyperparameter tuning and, later, feature-coin trajectory projection in the forecasting phase. Configurable arguments include the destination directory, output file names, path to the dataset, selected price variable, and the ratio for the train/validation split (expressed as floats).

Model pretrain (Fig. 6.1-❺) trains ML models (RNNs and GBMs) for cryptocoin forecasting. Its execution produces a pretrained model (Fig. 6.1-❻), stored either as a `.pth` file for PyTorch models (LSTM, GRU) or as a `.txt` file for GBMs (XGBoost, LightGBM, CatBoost). Forecasts for a target coin are generated from the trajectories of correlated feature coins, which can be pre-selected through correlation analysis. Configuration files in `.json` format specify model settings: for RNNs, the number of hidden layers, neurons per layer, training epochs, and batch size; for GBMs, the number of tree splits. Common parameters include learning rate, random seeds, and early-stopping patience. Configurable arguments include the destination directory, output file name, paths to train and validation sets, chosen ML model, target coin, feature-coin list, and configuration file.

Model forecast (Fig. 6.1-❼) generates price predictions using pretrained models. In this phase, the validation set is used to fit a Holt-Winters exponential smoothing model [224, 225], which projects feature-coin trajectories for the desired forecasting horizon. These predicted feature series are then passed to the pretrained ML model to produce forecasts for the target coin. Outputs are stored in `.txt` format. Configurable arguments include the destination directory, output file name, forecasting horizon (in days), path to the validation set, path to the pretrained model, chosen ML model, target coin, and feature-coin list. All arguments must remain consistent with those used in the pretraining phase.

Correlation analysis is optional but useful for feature selection. It computes correlations among cryptocoin prices using Pearson [205], Kendall [226], or Spearman [227] coefficients. Configurable arguments include the destination directory, output file names, path to the dataset, selected price variable, time window (daily, weekly, or monthly), and correlation method. This functionality guides users in choosing the most informative predictors for model training.

6.4. Illustrative Example

This section presents a walkthrough of the full forecasting pipeline with the CryptoAnalytics toolkit. File names and directories follow the structure in [59].

6.4.1. Data Pull

We begin by pulling OHLC market prices from CoinMarketCap. In this example, 8 cryptocoins are selected: Bitcoin (BTC), Ether (ETH), Tether (USDT), USD Coin (USDC), Ripple (XRP), Binance USD (BUSD), Cardano (ADA), and Dogecoin (DOGE), defined in the `coins.json` file inside the `/examples` directory. The time frame spans three months (15-08-2023 to 15-11-2023). The corresponding command (Fig. 6.1-❶) is:

```
python data_pull.py \
  --filename "dataset" \
  --coins "examples/coins.json" \
  --start "15-08-2023" \
  --end "15-11-2023"
```

this produces `dataset.csv` (Fig. 6.1-❷), consisting of six columns: `Date`, `Open`, `High`, `Low`, `Close`, and `Coin`.

6.4.2. Data Split

The dataset is then partitioned into training and validation subsets. Here, 80% of the data is used for training and 20% for validation, with the average OHLC price as the target variable. The corresponding command (Fig. 6.1-❸) is:

```
python data_split.py \
  --filenames "train" "valid" \
  --data "dataset.csv" \
  --variable "avg_ohlc" \
  --train 0.8 \
  --valid 0.2
```

this generates `train.csv`, from 15-08-2023 to 28-10-2023, and `valid.csv`, from 29-10-2023 to 15-11-2023 (Fig. 6.1-❹).

6.4.3. Correlation Analysis (Optional)

Correlation analysis can assist in selecting feature coins. Using BTC as the target, we identify five highly correlated assets (Pearson > 0.5) with a daily sliding window. The corresponding command is:

```
python correlation_analysis.py \
  --filename "correlations" \
  --data "dataset.csv" \
  --window "daily" \
  --method "pearson"
```

this produces `correlations.csv`, a cross-correlation matrix of all selected coins. Stablecoins are excluded, leaving ETH, XRP, DOGE, and ADA as feature variables.

6.4.4. Model Pretrain

Next, we pretrain an ML model to forecast the average OHLC price of Bitcoin. For this demonstration, we use an LSTM network with configurations defined in `config_nn.json` and the feature coins listed in `features.json`. The corresponding command (Fig. 6.1-❺) is:

```
python model_pretrain.py \
  --filename "lstm" \
  --train "train.csv" \
  --valid "valid.csv" \
  --target "btc" \
  --features "examples/features.json" \
  --model "lstm" \
  --config "examples/config_nn.json"
```

this creates a pretrained model stored as `lstm.pth` (Fig. 6.1-❻).

6.4.5. Model Forecast

Finally, we use the pretrained model to generate predictions on unseen data. Here, the goal is to forecast Bitcoin prices for the next 7 days. The corresponding command (Fig. 6.1-❼) is:

```
python model_forecast.py \
  --filename "predictions" \
  --valid "valid.csv" \
  --horizon 7 \
  --pretrained "lstm.pth" \
  --target "btc" \
  --features "examples/features.json" \
  --model "lstm"
```

this produces `predictions.txt`, containing forecasts of Bitcoin average OHLC prices for 16-11-2023 to 22-11-2023.

The forecasts achieved good accuracy, with MAPE $\approx 6.57\%$[1] and RMSE ≈ 2438.37 USD compared to observed CoinMarketCap prices.

```
Predicted: 34368.3, Real: 36878.7
Predicted: 34363.8, Real: 36341.8
Predicted: 34366.0, Real: 36570.9
```

[1]According to [228], a MAPE below 10% indicates a highly accurate model.

```
Predicted: 34368.5, Real: 36974.1
Predicted: 34365.5, Real: 37372.6
Predicted: 34368.4, Real: 36682.0
Predicted: 34364.3, Real: 36679.2
```

6.5. Deployment of CryptoAnalytics

This section discusses the deployment of CryptoAnalytics to build fast and reliable cryptocoin prediction services. We benchmarked the toolkit against three widely adopted serving frameworks: TorchServe [47], BentoML [48], and MLflow [49]. For MLflow, we considered two scenarios: a base setup with a local Flask server [177], and an alternative configuration with MLServer [50].

Experiments were conducted on a dedicated server-class machine running Ubuntu 22.04.2 LTS (Linux kernel 5.15.0-88-generic), equipped with a 64-core Intel Xeon® E5-2683 v4 CPU at 2.10 GHz and 128 GB RAM. Performance was evaluated using oha [199], which repeatedly submitted prediction requests to the CryptoAnalytics server over a two-minute interval. We measured response latency while varying the forecasting horizon (*i.e.*, the number of future daily prices to predict) from 0 (baseline) to 32 steps.

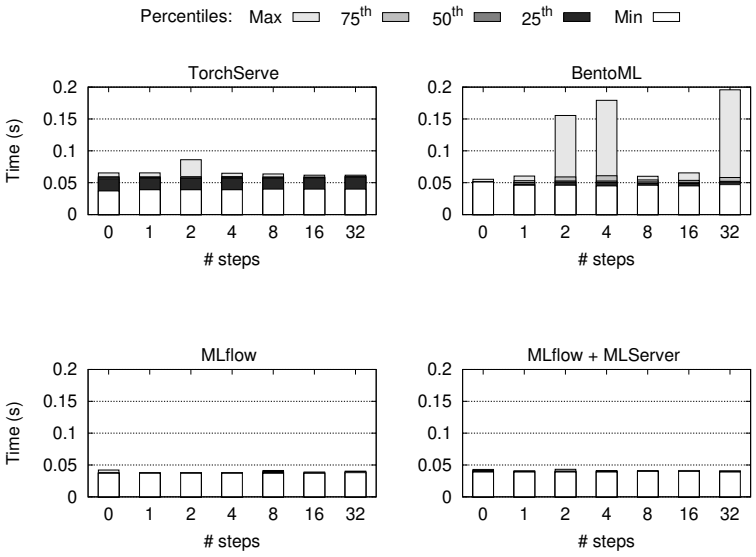

Figure 6.2.: Benchmark of deployment frameworks for CryptoAnalytics.

The results (Fig. 6.2) indicate that the forecasting horizon has negligible impact on latency. Among the tested frameworks, MLflow achieved the best performance, with delays consistently clustering around ≈ 0.04 seconds in both scenarios. BentoML showed higher latencies, while TorchServe reached intermediate levels.

These findings complement those of Chapter 4, which considered a wider range of workloads and metrics and found domain-specific frameworks such as TorchServe to be more efficient

overall. Together, the two perspectives suggest that MLflow provides a practical baseline for deploying CryptoAnalytics, while specialized frameworks remain preferable for large-scale scenarios where low-latency inference is critical.

Ready-to-use deployment configurations for all frameworks are available in [59].

6.6. Lessons Learned

The development and evaluation of CryptoAnalytics yield three main lessons.

First, although cryptocoin markets remain highly volatile, their price dynamics are not entirely random: persistent co-movement and correlation patterns can be systematically exploited for forecasting. This insight, demonstrated empirically in Chapter 5, underpins the design of CryptoAnalytics as a practical implementation of correlation-driven modeling.

Second, while advanced ML methods such as GBMs and RNNs enable accurate forecasts, their adoption is often hindered by the complexity of model configuration, training, and validation. CryptoAnalytics addresses this challenge by encapsulating statistical workflows into a streamlined CLI, lowering the entry barrier for non-experts while saving time for experienced practitioners.

Third, deployment experiments confirm that CryptoAnalytics integrates smoothly with serving frameworks including TorchServe, BentoML, and MLflow, enabling the construction of efficient and reliable forecasting services. This demonstrates the toolkit's versatility, both as a research instrument and as a foundation for production-grade applications.

6.7. Summary and Next Steps

This chapter introduced CryptoAnalytics, a Python-based toolkit for cryptocoin price forecasting. The toolkit automates the full pipeline, from data collection and preprocessing to model training, validation, and inference, with optional correlation analysis to guide feature selection. It supports both GBMs and RNNs, and its modular design enables seamless deployment with modern serving frameworks.

The lessons learned show that CryptoAnalytics serves both research and practice: researchers benefit from reproducibility and extensibility, while investors, institutions, and regulators gain a practical instrument for navigating volatile markets.

Looking ahead, several directions stand out. First, we plan to expand the range of supported data sources and models, further broadening applicability. Second, integration with streaming architectures could enable near real-time forecasting, extending the toolkit's value to high-frequency trading and risk monitoring. Finally, beyond cryptocoins, the general workflow of CryptoAnalytics can be adapted to domains where time-series co-movements and correlations play a central role, including finance, healthcare, and environmental monitoring.

By bridging advanced ML methods with accessible software design, CryptoAnalytics translates the theoretical insights of Chapter 5 into a concrete and reusable artifact, contributing both to reproducible research and to practical forecasting services.

Part IV.

Machine Learning for Smart Contract Security

Chapter 7.

PhishingHook: Phishing Detection in EVM Smart Contracts

In this chapter, we introduce PhishingHook, the first framework for detecting phishing Smart Contracts on the Ethereum blockchain through opcode-level analysis. We design a fully reproducible pipeline encompassing dataset construction, bytecode disassembly, model training, and post hoc interpretability. Sixteen Machine Learning models are benchmarked across four families, showing that static opcode-based detection achieves high accuracy without relying on transaction data. The results establish PhishingHook as a practical foundation for reproducible research and large-scale defense against phishing in decentralized ecosystems.

Chapter Outline

7.1. Introduction

Ethereum is one of the most widely used blockchains, second only to Bitcoin in market capitalization [229], and hosts a rapidly expanding ecosystem of Decentralized Applications (dApps) powered by Smart Contracts (SCs) [18]. These programs execute within the Ethereum Virtual Machine (EVM), a stack-based computing environment whose behavior can be formally verified [230] and whose semantics are defined through low-level opcodes governing arithmetic, memory, and control operations. The expressive power of SCs, combined with the large financial assets they manage, has made Ethereum a prime target of cyberattacks [25–28].

While vulnerabilities such as reentrancy flaws, arithmetic overflows, and front-running have been extensively studied, phishing attacks have recently become one of the most prevalent and damaging threats in the Ethereum ecosystem [231]. Unlike classical web-based phishing [232, 233], contract-level phishing leverages the programmability of the EVM to deceive users and exfiltrate assets. The scale and velocity of these attacks demand early and accurate detection, yet most existing solutions are proprietary [234], costly, and rarely reproducible. Moreover, the lack of open-source frameworks for benchmarking detection techniques leaves both researchers and practitioners without a clear basis for comparison.

To address this gap, we introduce PhishingHook [55, 56], a framework for detecting phishing SCs through opcode-level analysis. PhishingHook is the first system explicitly designed to support reproducible evaluation of diverse approaches, ranging from traditional Machine Learning (ML) techniques to modern Deep Learning (DL) and Large Language Models (LLMs). In particular, we benchmark 16 models, including seven architectures: two variants of Vision Transformers (ViTs), ESCORT [235], and two variants each of Generative Pretrained Transformer 2 (GPT-2) [108] and Text-to-Text Transfer Transformer (T5) [109], that have not previously been tested in this context. Beyond model benchmarking, the framework includes modules for dataset construction, bytecode extraction and disassembly, training and evaluation, and post hoc interpretability, making it a comprehensive environment for both research and deployment.

The contributions of this chapter are as follows:

- The construction and public release of the largest dataset of phishing SCs on Ethereum, available at [61];
- The design and implementation of the PhishingHook framework, streamlining data gathering, disassembly, and model evaluation;
- An extensive experimental study of 16 detection techniques, spanning traditional ML models, computer vision methods, LLMs, and specialized vulnerability detectors;
- A post hoc analysis offering opcode-level explainability for model predictions, supporting transparency and trust in automated detection.

By making datasets, source code, and workflows publicly available, PhishingHook advances reproducibility in blockchain security research and provides a practical foundation for building deployable phishing detection services.

7.2. Related Work

To the best of our knowledge, PhishingHook is the first framework dedicated exclusively to the detection of phishing SCs through opcode-level analysis. Prior research can be grouped into four main categories.

A first line of work addresses opcode-based fraud detection. These systems operate directly on contract bytecode to identify fraudulent behaviors such as Ponzi schemes or honeypots, though not phishing specifically. Examples include SCSGuard [147], which combines opcode n-grams with a Gated Recurrent Unit (GRU) and attention mechanism, and ECA+EfficientNet [148], which maps bytecode to RGB images for vision-based classification. HoneyBadger [138] uses symbolic execution and heuristics to detect honeypots, while Al-SPSD [139] applies ordered boosting over opcode features to flag Ponzi contracts. Other studies [236, 237] rely on supervised classifiers such as Random Forests (RFs) and K-Nearest Neighbors (KNN) after feature extraction from opcode sequences.

A second strand investigates transaction-based phishing detection. Rather than analyzing bytecode, these approaches examine transaction traces. Eth-PSD [140], for example, applies ML classifiers to scam-related transactions, while TxPhishScope [141] dynamically interacts with suspicious websites, triggering transactions to uncover associated phishing accounts. While effective at labeling transactions, such methods face scalability challenges for contract-level classification and may expose sensitive data when replaying malicious interactions. By contrast, PhishingHook focuses solely on contract code, thereby avoiding these limitations.

A third category concerns opcode-based vulnerability detection, where the goal is to uncover exploitable flaws rather than social engineering attacks. Systems such as ESCORT [235], CodeNet [143], and WIDENNET [144] are representative DL models for classifying vulnerable contracts. Similarly, BLSTM-ATT [145] and related BiLSTM architectures [146] target specific vulnerabilities like reentrancy bugs. While methodologically close in their reliance on opcodes, these approaches are not designed to capture phishing behaviors.

Finally, symbolic execution and verification tools provide formal or semi-formal contract analysis. DefectChecker [238] detects eight classes of defects, while Mythril [239], Securify2 [240], and Slither [241] are widely used verification frameworks. Comparative studies [242, 243] show their limited effectiveness, especially for complex contracts. Echidna [244] extends this line with fuzzing-based testing. Although effective at uncovering bugs, these approaches are computationally expensive and not tailored to phishing detection.

In summary, while prior work spans fraud detection, transaction analysis, vulnerability discovery, and formal verification, none provide an open, opcode-level, phishing-focused detection framework. PhishingHook fills this gap by systematically benchmarking a broad spectrum of models, enabling reproducible comparisons and advancing the defense against phishing in Ethereum.

7.3. The PhishingHook Framework

The architecture of PhishingHook consists of six building blocks: an initial data gathering phase, followed by the Bytecode Extraction Module (BEM), the Dataset Construction step, the Bytecode Disassembler Module (BDM), the Model Evaluation Module (MEM), and finally the Post Hoc Analysis Module (PAM). An overview is provided in Fig. 7.1.

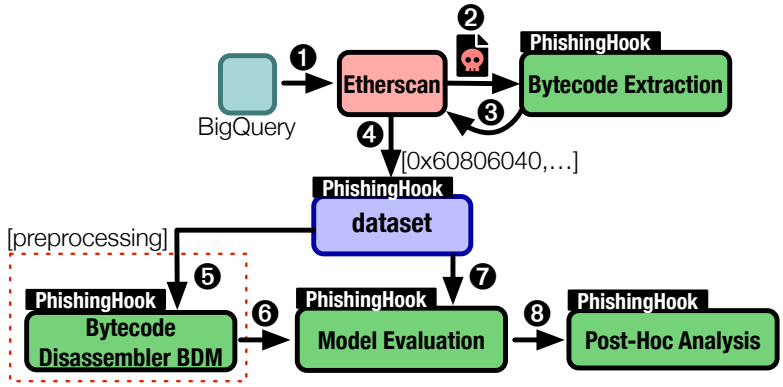

Figure 7.1.: The PhishingHook framework.

7.3.1. Data Gathering

The first step of PhishingHook collects a large pool of candidate SCs from the Ethereum blockchain. We rely on the public Ethereum dataset available through Google BigQuery [245], from which we retrieve a raw list of unlabeled contract hashes (Fig. 7.1-❶). For our study, we restrict the search to contracts deployed between October 2023 and October 2024, yielding roughly 4,000,000 candidates out of the 68,681,183 contracts publicly available as of October 22, 2024.

To label phishing SCs, we use etherscan.io [246], which flags malicious contracts with the label `Phish/Hack` (Fig. 7.1-❷). While any third-party labeling source can be incomplete and may introduce bias (*e.g.*, toward widely reported scams or known phishing patterns), Etherscan provides a structured reporting workflow in which suspicious addresses are first reported and then reviewed by the platform before a public label is applied [247]. Compared to purely community-driven repositories, this additional verification step helps reduce noise and adversarial labeling. Nevertheless, our dataset should be interpreted as capturing *confirmed* phishing contracts according to Etherscan at collection time, and it may under-represent undetected or emerging phishing campaigns.

We deliberately avoid fully community-based sources such as ChainAbuse [231], which have been shown to introduce bias in malicious contract labeling [248].

7.3.2. Bytecode Extraction Module

The Bytecode Extraction Module (BEM) is the first operational component of the framework. Given a set of labeled contract addresses, BEM queries etherscan using the `eth_getCode` endpoint via the JSON-RPC Application Programming Interface (API) to retrieve deployed bytecode (Fig. 7.1-❸). The resulting bytecodes form the core dataset used for model training and evaluation (Fig. 7.1-❹). This step is critical since bytecode is always publicly available on Ethereum, whereas source code is accessible only if voluntarily published.

7.3.3. Dataset Construction

From the initial 4 million collected contracts, we obtained 17,455 labeled as phishing. Among these, 3,458 correspond to unique bytecodes, while the remainder are duplicates. This high redundancy is largely due to minimal proxy contracts [249], *i.e.*, lightweight clones that share the same bytecode as their target contract. To balance the dataset, we added a comparable number of benign samples (contracts not flagged as malicious). The final dataset includes 7,000 bytecodes, released publicly [61]. To our knowledge, this is the largest dataset of phishing SCs currently available. The temporal distribution of phishing contracts over the observation period is shown in Fig. 7.2. The pronounced increase observed between January and March 2024 is consistent with external reports documenting a sharp rise in phishing activity during this period, particularly linked to the rapid growth of speculative trading activity [250].

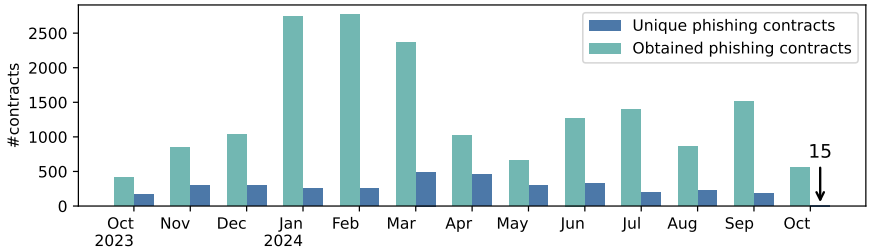

Figure 7.2.: Number of phishing contracts per month from October 2023 to October 2024.

7.3.4. Bytecode Disassembler Module

The Bytecode Disassembler Module (BDM) converts raw bytecode into sequences of opcodes (Fig. 7.1-❺). This step is essential since malicious actors rarely release source code, making static analysis dependent on bytecode. BDM outputs each instruction with its mnemonic (human-readable alias), operand, and gas cost. For instance, the bytecode `0x6080604052` is disassembled into the sequence: `(PUSH1, 0x80, 3), (PUSH1, 0x40, 3), (MSTORE, NaN, 3)`. Results are stored in `.csv` format for downstream tasks (Fig. 7.1-❻).

In practice, BDM is required only for a subset of models, namely Histogram Similarity Classifiers (HSCs) and ViT+Frequency, which operate on opcode-level features rather than raw bytecode. To implement this, we extended the Python library evmdasm [251], originally last

updated during the Arrow Glacier fork (2022). Our extension adds support for new EVM instructions introduced in the Shanghai fork, including `INVALID` and `PUSH0`. This enhanced version is released alongside our dataset [61].

A natural question is whether opcode frequency alone suffices to distinguish phishing from benign contracts. As shown in Fig. 7.3, the distribution of opcode usage across 20 representative instructions is similar for both classes, indicating that phishing cannot be reliably detected by frequency-based heuristics alone.

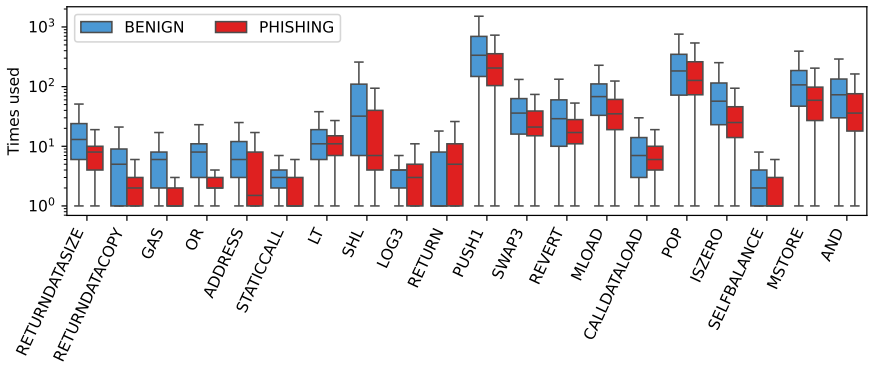

Figure 7.3.: Opcode usage distribution for 20 selected instructions across phishing and benign contracts.

7.3.5. Model Evaluation Module

The Model Evaluation Module (MEM) (Fig. 7.1-❼) performs systematic training and benchmarking of ML models for phishing detection. We evaluate 16 models across four categories: *(i)* Histogram Similarity Classifiers (HSCs), *(ii)* Computer Vision Models (CVMs), *(iii)* Large Language Models (LLMs), and *(iv)* Vulnerability Detection Models (VDMs). This includes state-of-the-art methods originally designed for fraud detection in SCs (*e.g.*, ECA+EfficientNet, SCSGuard) as well as modern architectures adapted from other domains, such as ViT, GPT-2, and T5. Transformers are preferred over Recurrent Neural Networks (RNNs) given their superior representation learning capabilities for sequential code and malware detection [252, 253]. Model configurations and evaluation metrics are detailed in Section 7.4.

7.3.6. Post Hoc Analysis Module

The Post Hoc Analysis Module (PAM) (Fig. 7.1-❽) provides statistical validation of the results obtained in MEM. Its role is to ensure that performance differences across models are statistically significant rather than due to random variation. We rely on standard statistical tests and post hoc procedures, implemented in R, to compare families of models. PAM thus complements MEM by grounding conclusions in rigorous statistical analysis. Further details on the procedures are provided in Section 7.4.5.

7.4. Evaluation

This section evaluates the performance of PhishingHook across multiple detection models and experimental settings. We describe the hardware and software environment, present the benchmarked models, and discuss the statistical procedures used to compare results.

7.4.1. Experimental Setup

The evaluation was performed on two GPU-enabled nodes. The first machine features an Intel Core i7-14700KF® CPU (28 cores, 5.5 GHz), 64 GiB of RAM, and an NVIDIA GeForce RTX® 4090 GPU with 24 GiB of VRAM, running CUDA 12.2. The second machine is equipped with an Intel Xeon Platinum 8562Y+® CPU (32 cores, 2.8 GHz), 126 GiB of RAM, and an NVIDIA® H100 NVL GPU with 94 GiB of VRAM, running CUDA 12.4.

For DL models, training and testing were implemented in PyTorch v2.5, while classical ML models were developed using Scikit-learn v1.5. The post hoc analysis module of PhishingHook was implemented in R v4.4.

7.4.2. Compared Models

PhishingHook addresses a binary classification task: determining whether a SC is a phishing contract or not. To this end, we evaluate a broad range of detection approaches drawn from different domains, reimplemented from scratch within the PhishingHook framework, since original implementations were not publicly available. In total, 16 models are benchmarked, spanning Histogram Similarity Classifiers (HSCs), Computer Vision Models (CVMs), Large Language Models (LLMs), and a Vulnerability Detection Model (VDM). They are summarized below.

HSCs. Following [236], each contract bytecode is represented as a histogram of opcode occurrences, yielding a vector of length equal to the number of unique opcodes in the training set. This vector is directly fed, without normalization or standardization, into several classical ML classifiers from Scikit-learn [176]: Random Forest (RF), Light Gradient Boosting Machine (LightGBM), K-Nearest Neighbors (KNN), Extreme Gradient Boosting (XGBoost), Categorical Boosting (CatBoost), Logistic Regression (LogReg), and Support Vector Machine (SVM).

ViT+R2D2 (CVM). Inspired by Android malware detection research [254], bytecodes are interpreted as sequences of hexadecimal color codes mapped into the RGB space. The resulting pixels form a $224 \times 224 \times 3$ tensor (zero-padded if necessary), which is processed by a Vision Transformer (ViT-B/16) [107] pretrained on ImageNet [191, 198] (weights from Hugging Face [255]) and fine-tuned on the phishing classification task.

ViT+Freq (CVM). Each opcode and operand from disassembled bytecode is encoded into a frequency value based on its occurrence in the training set, mapped into RGB channel intensities. The resulting fixed-size $224 \times 224 \times 3$ tensor (zero-padded if required) is processed by the same pretrained ViT-B/16 model as in ViT+R2D2. This follows the frequency encoding approach proposed in [256].

ECA+EfficientNet (CVM). Following [148], contract bytecodes are mapped into RGB images and classified using an EfficientNet-B0 backbone [100] augmented with an Efficient Channel Attention (ECA) module [101]. A global average pooling layer reduces dimensionality before classification.

SCSGuard (LLM). Proposed by [147], SCSGuard encodes bytecode as n-grams (6-character hexadecimal strings) mapped into integer indices and padded to uniform length. The model combines an embedding layer, a multi-head attention block, a GRU layer to capture sequential dependencies, and a fully connected classification head.

GPT-2 and T5 (LLMs). We fine-tune two Transformer-based language models, GPT-2 [108] and T5 [109], using Hugging Face implementations [172]. Although originally designed for text, these models can process raw bytecode sequences tokenized via the `GPT2Tokenizer` and `T5Tokenizer`. Despite their moderate size (up to 1.5B parameters), such models have shown strong performance in classification tasks [257, 258].

ESCORT (VDM). Originally introduced for SC vulnerability detection [235], ESCORT embeds contract bytecodes into a vector space processed by a deep neural network. While its primary purpose is multi-class vulnerability classification with transfer learning capabilities, here it is repurposed for binary phishing detection. To the best of our knowledge, this is the first application of ESCORT to phishing SCs.

7.4.3. Hyperparameter Search

We use Optuna [259], an open-source framework for automated hyperparameter optimization, to tune all models in PhishingHook. Optuna employs metaheuristic search strategies and a define-by-run API, which enables dynamic construction of search spaces at runtime. For each model, optimization was performed through grid search over the defined parameter space, using 10-fold cross-validation to ensure robustness and prevent overfitting.

7.4.4. Analysis of Results

To ensure the stability of results, we performed a 10-fold cross-validation over three runs, for a total of 30 experiments per model (excluding hyperparameter optimization). The metrics reported in Table 7.1 are averaged over all folds and runs.

We evaluate two variants of GPT-2 and T5: α, where opcode sequences are truncated to fit model token limits and GPU constraints (NVIDIA GeForce RTX® 4090), and β, trained on an NVIDIA® H100 NVL GPU, where full bytecodes are processed in chunks using a sliding window. SCSGuard, based on n-gram encoding, remains unaffected by these constraints. Our results show that the vulnerability detector ESCORT is not effective when adapted to the classification of phishing SCs. This limitation stems from the intrinsic mismatch between vulnerability detection and phishing, as the latter exploits behavioral deception patterns rather than explicit technical flaws in contract logic. By contrast, vision-based models operating on bytecode images can still perform well, as they capture localized statistical regularities in opcode distributions without relying on explicit semantic understanding of vulnerabilities. This distinction is further

analyzed through post hoc interpretability in Section 8.5.6, where class activation mapping reveals that vision models focus on compact, early bytecode regions rather than full control-flow semantics.

The most accurate models are the HSCs, with an average Accuracy of 91.52%, F1 Score of 91.44%, Precision of 91.61%, and Recall of 91.32%. Among them, the RF classifier achieves the best overall performance. LLMs rank second, with an average Accuracy of 88.83%, F1 Score of 88.17%, Precision of 89.50%, and Recall of 88.07%, where SCSGuard is the strongest performer. CVMs report an average Accuracy of 83.75%, F1 Score of 83.40%, Precision of 83.26%, and Recall of 83.63%, with ECA+EfficientNet outperforming other vision-based methods. Overall, all models achieve reasonable accuracy in distinguishing phishing from benign contracts, with a global average of 89.07% Accuracy, 88.74% F1, 89.24% Precision, and 88.70% Recall.

These findings are consistent with prior studies. For HSCs, RF also emerged as the top model in [236], though with slightly lower Accuracy (85.17%). In contrast, ECA+EfficientNet [148] reported 98.2% Accuracy, but on broader fraud datasets, making direct comparison to our phishing-specific results difficult. SCSGuard [147] correctly detected four out of five phishing scams, with a lower overall Accuracy of 80%. The remaining models were newly implemented within PhishingHook; ESCORT, originally designed for vulnerability detection, had not been tested for fraud classification before this study.

The reported metrics are essential for evaluating the effectiveness of malware and fraud detection systems [260]. However, performance must also be weighed against computational cost, including training and inference times relative to dataset size, an aspect examined in Section 7.4.6.

7.4.5. Post Hoc Analysis

This section describes the methodologies applied for the post hoc analysis of results. The objective is to rigorously compare model performance metrics (Accuracy, Precision, Recall, and F1 Score) and assess the statistical significance of observed differences.

The vulnerability detector ESCORT was excluded from this analysis due to its poor performance on the phishing detection task. Similarly, GPT-2_β and T5_β were omitted as the lowest-performing variants of their respective models. The post hoc analysis was conducted on the complete experimental results, comprising 30 trials per model (10-fold cross-validation $\times$ 3 runs), for a total of 390 observations (13 models $\times$ 30 trials).

We first tested the normality of each metric distribution using the Shapiro-Wilk (S-W) test [261]. This step is essential, since the subsequent choice between parametric and non-parametric tests depends on whether normality holds. The S-W statistic W, indicating the likelihood of observing the data under the null hypothesis, is computed as:

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Table 7.1.: Performance metrics (%) for models supported in PhishingHook. Best values are shown in bold. Symbols: †: Histogram, ‡: Vision, ∗: Language, §: Vulnerability.

Model	Accuracy	F1 Score	Precision	Recall
Random Forest†	**93.63**	**93.49**	94.23	92.76
k-NN†	90.60	90.62	89.31	91.99
SVM†	92.60	92.32	**94.53**	90.21
Logistic Regression†	83.91	84.13	82.03	86.38
XGBoost†	93.43	93.30	93.74	**92.88**
LightGBM†	93.39	93.26	93.80	92.73
CatBoost†	93.10	92.95	93.62	92.30
ECA+EfficientNet‡	86.63	86.16	86.88	85.52
ViT+R2D2‡	85.52	85.14	85.20	85.15
ViT+Freq‡	79.11	78.90	77.71	80.23
SCSGuard*	90.46	90.12	90.95	89.35
GPT-2^*_α	89.95	89.60	90.39	88.91
$T5^*_\alpha$	89.67	89.28	90.25	88.35
GPT-2^*_β	88.65	88.36	88.40	88.36
$T5^*_\beta$	85.41	83.47	87.49	85.40
ESCORT§	55.91	55.82	55.78	55.91

Table 7.2.: Results of the Kruskal-Wallis test for performance metrics. Significant if $p_{adj} < 0.05$.

Metric	H	p	p_{adj}
Accuracy	360.81	7.35×10^{-70}	2.94×10^{-69}
F1 Score	359.78	1.21×10^{-69}	3.63×10^{-69}
Precision	345.21	1.44×10^{-66}	2.88×10^{-66}
Recall	322.03	1.10×10^{-61}	1.10×10^{-61}

where $x_{(i)}$ are ordered sample observations, a_i are coefficients, $\bar{x}$ is the sample mean, and n is the sample size. The null hypothesis of normality is rejected for low W values ($p < 0.05$). In our case, normality was violated for 20 out of 52 model-metric pairs. Consequently, we applied the non-parametric Kruskal-Wallis (K-W) test [262] to determine whether statistically significant differences exist among model medians.

The K-W statistic H is defined as:

$$H = \frac{12}{N(N+1)} \sum_{i=1}^{k} \frac{R_i^2}{n_i} - 3(N+1)$$

where k is the number of groups, n_i the number of observations in group i, N the total number of observations, and R_i the sum of ranks within group i.

The null hypothesis that all models share the same median performance is rejected for high H values ($p < 0.05$). To mitigate false positives, we applied the Holm-Bonferroni correction [263].

As shown in Table 7.2, the null hypothesis was rejected for all four metrics, confirming statistically significant performance differences across models.

To identify which model pairs differed, we performed a Dunn's test [264] with Holm-Bonferroni correction, the appropriate non-parametric post hoc procedure following a rejected Kruskal-Wallis test [265]. The Dunn's test statistic Z is computed as:

$$Z = \frac{(\bar{R}_i - \bar{R}_j)}{\sqrt{\frac{N(N+1)}{12}\left(\frac{1}{n_i} + \frac{1}{n_j}\right)}}$$

where $(\bar{R}_i - \bar{R}_j)$ is the difference in mean ranks between groups i and j, and the denominator accounts for rank variance adjusted by group size.

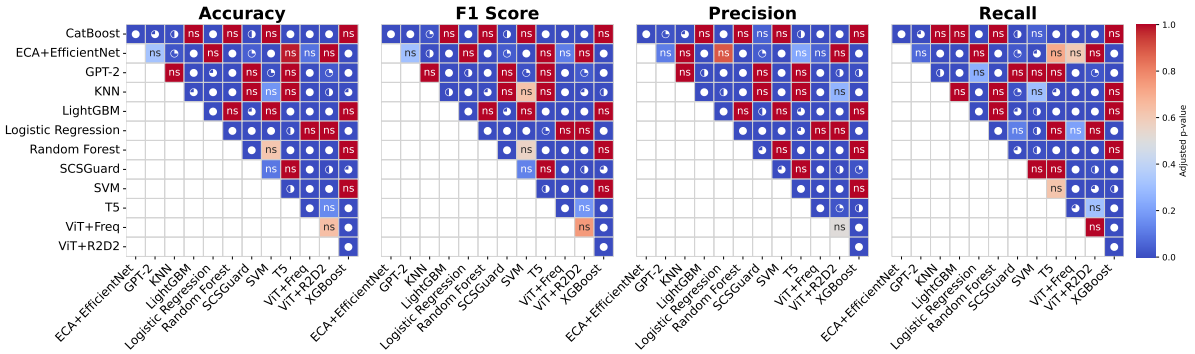

Figure 7.4.: Dunn's test for pairwise comparisons between model metrics. Significant if $p_{\text{adj}} < 0.05$. Significance levels range from ● (highly significant) to ◔ (mildly significant). Non-significant results are labeled as **ns**.

Figure 7.4 summarizes the results. For Accuracy, F1 Score, and Precision, 65.38% of model pairs exhibited statistically significant performance differences, while for Recall the proportion was 61.54%. When restricting to pairs within the same model family (*e.g.*, SVM and k-NN for HSCs, GPT-2 and T5 for LLMs), the proportion of significant differences decreased: 37.04% for Accuracy and F1, 40.74% for Precision, and 33.33% for Recall. Conversely, for model pairs belonging to different categories, these values increased to 80.39%, 78.43%, and 76.47% respectively. Overall, the results confirm that differences are highly significant across heterogeneous model families, whereas intra-family variations are comparatively minor.

7.4.6. Model Scalability Analysis

We assess the scalability of the evaluated models to study how performance varies with dataset size. Three data splits were generated, containing respectively one-third, two-thirds, and all available SCs. We trained and tested three representative models, SCSGuard, ECA+EfficientNet, and RF, corresponding to the best-performing approaches within the LLM, CVM, and HSC families, respectively. Figure 7.5 summarizes the results. RF maintains the highest accuracy

across all splits, showing stable behavior regardless of dataset size. In contrast, both SCSGuard and ECA+EfficientNet exhibit clear improvements as the number of training samples increases, suggesting that larger datasets may enable complex architectures, particularly CVMs and LLMs, to eventually outperform HSCs.

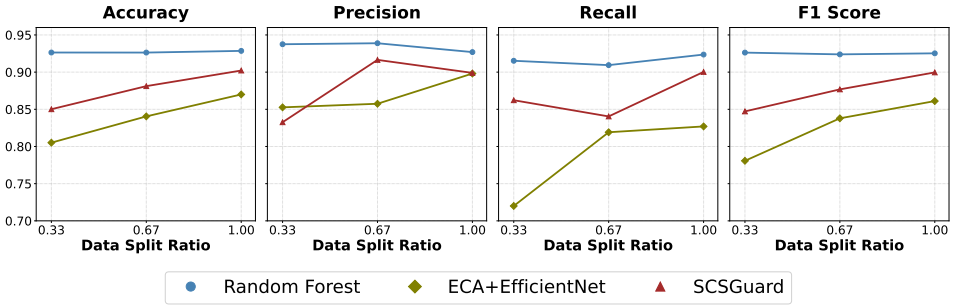

Figure 7.5.: Performance metrics of the best models across different dataset splits.

To further analyze model behavior under different dataset sizes, we employ a Critical Difference Diagram (CDD) [266], shown in Figure 7.6. The diagram provides a compact visualization of post hoc comparisons. First, a Friedman test [267] identifies whether performance differences among models are statistically significant across data splits. When differences are found, a Wilcoxon signed-rank test [268] determines which model pairs diverge significantly. In the diagram, rightmost elements correspond to the best-performing models (RF for all metrics), while leftmost ones denote weaker performers (ECA+EfficientNet). The thick horizontal bar connects classifiers that are not significantly different from each other. Across all metrics, p values were either 0.25 or 0.75, yielding adjusted $p_{adj} = 0.75$. Cliff's δ [269] indicates varying effect sizes, with notably negative values for SCSGuard compared to ECA+EfficientNet (-0.778 for Accuracy and F1 Score, -0.333 for Precision, and -1.0 for Recall), indicating performance declines. However, the high p_{adj} values imply that these differences are not statistically significant. This outcome is likely influenced by the limited sample size of the scalability experiment (36 measurements in total), as non-parametric methods require larger datasets for robust statistical power [270].

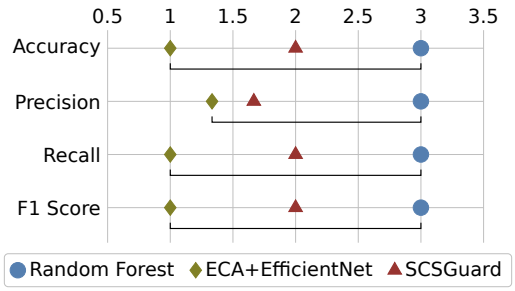

Figure 7.6.: Critical difference diagram of model scalability.

We also examined training and inference times for the three models, reported in Figure 7.7. Dataset size has a substantial effect on complex models, especially LLMs. On average across all

splits, SCSGuard requires 325.3 s of training time (+64,733.4% compared to RF and +1,030.6% compared to ECA+EfficientNet). Its average inference time per batch is 6.09 s (+57,258.5% over RF and +622.4% over ECA+EfficientNet). For SCSGuard, both training and inference times nearly double with each increase in dataset size (+77.35% and +68.8% on average, respectively), while HSCs and CVMs remain comparatively stable and fast.

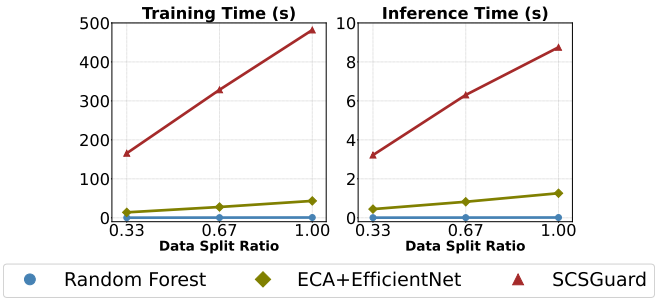

Figure 7.7.: Training and inference times for the best models across dataset splits.

From an operational perspective, these results indicate that inference time is acceptable or problematic depending on the deployment context rather than intrinsically good or bad. For real-time settings such as cryptocurrency wallets or browser-based transaction approvals, the low-latency behavior of HSCs makes them more suitable, as even small delays may expose users to financial risk. Conversely, in offline or semi-online scenarios, including token listing pipelines, compliance checks, and large-scale security audits, the higher inference cost of complex models may be acceptable given their improved scalability with data volume. Therefore, while larger datasets can enhance the accuracy of LLMs and CVMs, their inference overhead represents a practical trade-off rather than a categorical limitation.

7.4.7. Time-Resistance Analysis

Building on [271], we design a time-resistance experiment to evaluate the robustness of PhishingHook against temporal performance decay. A second dataset of 7,000 samples is constructed, ensuring that benign contracts follow the same temporal distribution as phishing ones (Figure 7.2). The training set includes contracts deployed between October 2023 and January 2024, while nine subsequent test sets, covering February to October 2024, are used to assess performance evolution over time.

As in the scalability experiment, three representative models are evaluated (SCSGuard, ECA+EfficientNet, and RF) to determine whether phishing contracts exhibit persistent opcode patterns or evolve to evade detection. The results, shown in Figure 7.8, reveal stable detection performance across months, with only a slight decline likely caused by evolving attacker behaviors, consistent with prior observations in [271].

To quantify stability over time, we adopt the Area Under Time (AUT) metric, defined as the area under the F1 score curve for phishing samples, with AUT $\in [0, 1]$. Higher AUT values indicate stronger robustness to temporal concept drift. RF achieves the highest stability (AUT = 0.89), followed by SCSGuard (0.84) and ECA+EfficientNet (0.79), the latter showing greater monthly

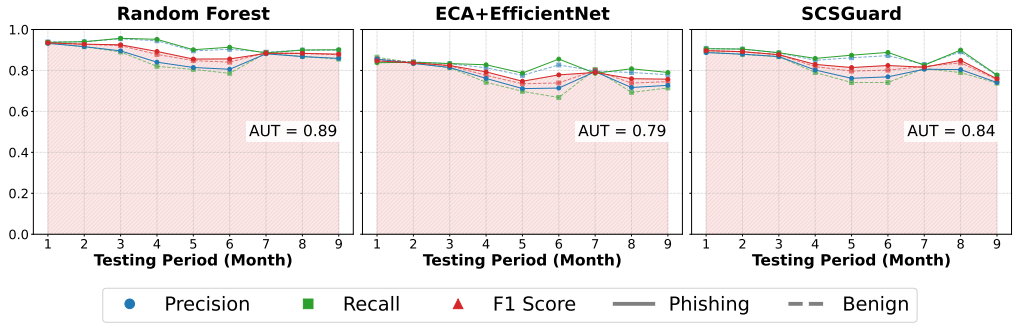

Figure 7.8.: Temporal evolution of performance metrics over nine months, with AUT computed for the phishing samples' F1 score.

variation. Despite these minor fluctuations, PhishingHook maintains consistent accuracy over time, demonstrating resilience to evolving attack strategies and confirming its suitability as a long-term phishing detection framework.

The dataset used for this temporal evaluation is publicly available at [61].

7.4.8. Opcode-Level Explainability

Beyond the statistical analyses discussed earlier, examining opcode influence scores provides complementary insights into model behavior and feature relevance. This section focuses on the best-performing classifier: the HSC model based on RF.

Figure 7.9 reports the Shapley values [272] computed for the 700 samples comprising the test set of a random fold from Section 7.4.4. Alternative interpretability methods, such as impurity-based feature importance, are also supported in PhishingHook, but they can introduce bias toward high-cardinality features and yield misleading importance scores [273]. In contrast, Shapley values provide an additive and model-agnostic explanation of each feature's contribution to individual predictions.

Figure 7.9 should be read as follows: opcodes at the top contribute most to the classifier's decision, while the horizontal spread indicates the direction and magnitude of their influence. Positive SHAP values increase the predicted probability of phishing, whereas negative values push the prediction toward benign. Importantly, the model does not rely on a single opcode but on a combination of weak signals whose joint distribution characterizes phishing behavior.

The vertical axis in Figure 7.9 represents the SHAP value, which quantifies how much each feature (here, opcode usage frequency) shifts the model's prediction away from the baseline phishing probability across all contracts. Each point corresponds to a single contract: its position along the horizontal axis reflects the feature value, while color encodes the magnitude of its effect.

For instance, the cluster of contracts with SHAP values around 0.025 showing low usage of the `GAS` opcode (third column) suggests that the classifier tends to associate reduced gas activity

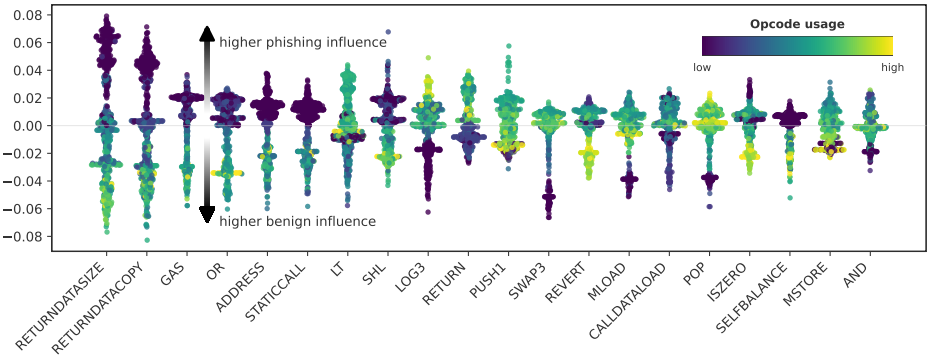

Figure 7.9.: SHAP values of the HSC classifier for all samples in a test split (top 20 most influential opcodes).

with a higher likelihood of phishing. In well-structured contracts, gas usage is typically handled explicitly and defensively, especially around external calls such as [274]:

```
Address.functionDelegateCall(address(this), data[i]);
```

this defensive behavior may involve explicit gas checks or structured call patterns, which do not necessarily translate into frequent occurrences of the `GAS` opcode itself at the bytecode level.

Phishing contracts, in contrast, often omit such checks, prioritizing rapid fund extraction over execution safety.

However, opcode-level patterns can sometimes be ambiguous. For example, the `GAS` instruction may appear in benign contracts within deprecated logic, such as [275]:

```
if (deprecated) {
    return UpgradedStandardToken(upgradedAddress)
        .transferByLegacy(msg.sender, _to, _value);
}
```

which produces similar disassembled bytecode. Such overlaps can reduce discriminative power and occasionally yield false negatives. This highlights that while opcode frequency supports valuable interpretability, effective phishing detection ultimately requires combining opcode semantics with higher-level execution context.

7.5. Lessons Learned

PhishingHook demonstrates that it is possible to build efficient and reproducible ML-based systems for detecting phishing activities directly from EVM bytecode. Our experiments show

that purely static opcode-level analysis, without relying on transaction data or dynamic node interactions, already achieves strong predictive performance, with accuracies around 90% across most models.

> Leveraging PhishingHook, highly effective opcode-based phishing detection systems can be built without dynamic data or transaction traces.

To mitigate challenges caused by duplicated minimal proxy contracts, we integrated a dedicated post hoc analysis module, allowing statistical generalization of findings from sampled data to the broader Ethereum contract population. This analysis revealed significant performance differences across model families, whereas variations within the same category were less pronounced.

> LLMs and CVMs do not outperform HSCs for phishing detection. Differences within the same model families are generally less significant.

Our scalability experiments further showed that while simpler models such as HSCs remain the most time-efficient, more complex architectures, particularly LLMs, exhibit superior scalability as data volume increases. This observation aligns with the well-established principle that large models often require more data to achieve strong generalization [276]. Consequently, expanding the dataset is likely to improve detection accuracy beyond the current ≈94% ceiling achieved by RF.

> Complex models, especially LLMs, scale better than HSCs. Larger datasets can further enhance overall accuracy.

Finally, the time-resistance analysis confirms that PhishingHook remains robust against evolving phishing strategies. HSCs display the most stable long-term performance, while LLMs and CVMs show minor fluctuations yet maintain consistent detection capabilities over time.

> PhishingHook is resilient to evolving threats. Despite minor temporal variations, it consistently maintains reliable long-term performance.

7.6. Summary and Next Steps

PhishingHook constitutes the first comprehensive framework dedicated to detecting phishing attacks in Ethereum SCs through opcode and bytecode-level analysis. Across sixteen evaluated models, our results confirm the strong effectiveness of opcode-based approaches, which consistently achieve high accuracy and stable performance across diverse model families.

This work contributes in several key directions: the design and implementation of the PhishingHook framework, the construction and public release of the largest dataset of phishing SCs

to date, and a detailed comparative evaluation of multiple detection techniques. The experiments show that while complex architectures such as LLMs do not necessarily outperform simpler classifiers, they scale more effectively as data volume increases, highlighting the trade-off between model complexity and data efficiency.

By releasing PhishingHook as an open and reproducible platform, we enable future research and development in blockchain and SC security. Our findings lay the groundwork for further advances in the detection of malicious behaviors within decentralized ecosystems.

Several directions for future work emerge from this study. First, a deeper analysis of misclassified contracts is needed to better understand model limitations. In particular, future investigations should examine whether false positives and false negatives are shared across different model families, whether they correspond to specific phishing strategies or contract templates, and how such contracts are structured and deployed in practice. This analysis could reveal systematic blind spots and guide the design of more robust hybrid detection strategies.

Second, we plan to extend PhishingHook toward real-time detection of phishing contracts before deployment on-chain and to integrate scalable inference pipelines suitable for production-grade monitoring. We also envision its adoption by cybersecurity firms and blockchain monitoring platforms such as Etherscan, strengthening proactive defense against contract-based phishing. Finally, the methodology underlying PhishingHook can be naturally extended to other blockchain threats, including Ponzi schemes and honeypot attacks.

Chapter 8.

Seeing Through EVM Bytecode Obfuscation

This chapter examines the resilience of Smart Contract malware detection in the presence of adversarial bytecode obfuscation. Although obfuscation is widely used to hinder reverse engineering, its practical impact on the security of Ethereum contracts, both in terms of syntactic validity and semantic preservation, remains poorly understood. By evaluating two state-of-the-art obfuscators across a large corpus of real-world phishing and honeypot malware, we assess how structural rewriting affects deployability, functional behavior, gas usage, and the performance of diverse Machine Learning detectors. This analysis provides the first empirical foundation for understanding how obfuscation reshapes the risks and defenses surrounding malicious Smart Contracts.

Chapter Outline

8.1. Introduction

SCs on Ethereum autonomously manage digital assets and govern dApps, operating within a financial landscape that routinely involves billions of dollars [25, 26]. These programs execute on the EVM, a stack-based runtime that interprets low-level bytecode composed of opcodes defining the contract's logic and control flow.

Malicious SCs have become a persistent threat across blockchain ecosystems. In October 2022, Solidus Labs [277] reported 188 525 scam contracts across twelve monitored blockchains, including Ethereum, Polygon [278], and BNB Chain [279], confirming both the scale and persistence of contract-level attacks.

To evade detection, adversaries increasingly apply obfuscation, both at the source level and directly at the bytecode level. Such transformations rewrite control flow and instruction layout while preserving deployability and intended malicious behavior. Tools such as BiAn [31] demonstrate that similar strategies can be injected into Solidity source code, complicating static inspection even before compilation.

Yet the practical impact of these transformations remains largely unexplored. It is unclear whether obfuscated malware preserves functionality, whether the resulting bytecode remains valid under EVM constraints, and to what extent existing ML-based detection models can still recognize such variants.

This chapter presents the first systematic evaluation of EVM bytecode obfuscation and its effects on SC malware detection. Using two open-source obfuscation tools, BOSC [32] and EVeilM [33], together with two real-world datasets (PhishingHook, presented in Chapter 7, containing 3,458 labeled phishing contracts [55, 56, 61], and HoneyBadger [138], comprising 546 honeypots), we generate 27 distinct obfuscation configurations applied to all malware samples. The resulting variants are evaluated along four dimensions: runtime validity, semantic preservation, gas usage, and detection effectiveness.

Our findings show that BOSC yields structurally valid bytecode in only 18-39% of cases (as measured by successful Control Flow Graph (CFG) reconstruction with Heimdall [280] and EVMLiSA [281] in Section 8.5.2). To assess whether obfuscation preserves malicious behavior, we additionally run a semantic equivalence check between each obfuscated sample and its original contract using symbolic execution with hevm [282] (Section 8.5.4). Under this criterion, BOSC rarely preserves behavior, whereas EVeilM maintains 84-100% structural validity but yields semantic equivalence for only 15-28% of samples.

Most detection approaches experience substantial degradation when exposed to obfuscated phishing contracts, though the CVM ECA+EfficientNet [148] remains comparatively robust across several configurations. These observations indicate that current obfuscation techniques often distort programs beyond deployability or semantic fidelity, and that vision-based representations may offer improved resilience against such transformations.

The contributions of this study [57] are as follows:

- a methodology for generating and validating 27 obfuscation configurations across BOSC and EVeilM;
- the first comprehensive evaluation of EVM bytecode obfuscation and its impact on malware detection;
- an assessment of eight ML-based detectors, including histogram, language, vision, and graph-based approaches, under obfuscation;
- the public release of code, models, configurations, and datasets at [62] to enable full reproducibility.

8.2. Related Work

Research on SC security spans static analysis, symbolic execution, and ML-based classification. However, existing approaches overwhelmingly assume access to non-obfuscated, structurally intact bytecode. This chapter examines a complementary and largely unexplored question: the robustness of malware detection under EVM bytecode obfuscation.

A first line of work applies ML models directly to runtime bytecode for malicious contract detection. PhishingHook (Chapter 7) provides the most extensive evaluation to date, benchmarking sixteen classifiers for phishing detection. Other studies focus on narrower fraud categories, including honeypots in HoneyBadger [138] and Ponzi schemes in Al-SPSD [139]. Classical models such as RFs, SVMs, and KNN have been applied to opcode-level features [236, 237], but only under clean, unobfuscated conditions.

A second strand uses behavioral and transaction-level analysis. Eth-PSD [140] and TxPhishScope [141] detect suspicious activity by inspecting execution traces and user interactions. While effective after deployment, such methods introduce latency and may raise privacy concerns, limiting their suitability as proactive defenses.

Representation-learning approaches have also emerged. ESCORT [142], CodeNet [143], and WIDENNET [144] learn embeddings from opcode or bytecode to capture semantic patterns, while attention-based recurrent models [145, 146] apply sequence modeling inspired by Natural Language Processings (NLPs). Although these methods improve performance on raw bytecode, none evaluate robustness when control flow or instruction layout is intentionally transformed.

Static analysis and symbolic execution tools are central to vulnerability detection. Mythril [239], Securify2 [240], Slither [241], and hevm [282] operate under well-formed bytecode assumptions and are primarily designed to uncover bugs rather than assess semantic preservation under program rewriting. No prior work has applied symbolic execution at scale to evaluate whether obfuscation maintains functional behavior.

Compared to PhishingHook, which evaluates detection on raw bytecode, this study measures robustness on obfuscated variants generated through 27 configurations. Unlike ESCORT [142], which analyzes deep embeddings on clean inputs, we assess their behavior when control flow is

adversarially modified. Relative to symbolic execution tools such as hevm, we use equivalence checking to quantify semantic preservation across thousands of transformed samples.

To date, no study has systematically examined:

- whether EVM obfuscation preserves bytecode validity or semantics at scale,
- how detection models from multiple families behave under real obfuscation pipelines, or
- which feature representations remain robust when structural regularities are removed.

This gap motivates the present evaluation, which provides the first large-scale analysis of EVMs bytecode obfuscation, quantifies semantic preservation using symbolic execution, and measures the robustness of diverse detection approaches under structural and behavioral transformations. The results establish an empirical foundation for developing obfuscation-resilient malware detectors in the Ethereum ecosystem.

8.3. Smart Contract Bytecode Obfuscation

Obfuscation is widely used to hinder automated analysis and manual auditing. Although often deployed to protect intellectual property, the same techniques can be misused to conceal malicious behavior. Source-level approaches such as BiAn [31] modify data and control flow structures directly in Solidity, but apply only to verified contracts [283], since access to the original source code is required. Because EVM bytecode is always stored on-chain, bytecode-level obfuscation remains applicable to unverified or proprietary contracts and therefore constitutes a broader attack surface.

Only a small number of tools currently support EVM bytecode obfuscation. BOSC [32] implements four transformations designed to disrupt static analysis and decompilation while preserving deployability and semantics in principle.

Incomplete inserts malformed or partially specified instructions into the bytecode. For example, starting from the original sample:

```
0x6080604052348015610010576000 80fd5b50...
```

BOSC injects the following malformed sequence:

```
a457c2d711610066578063a457c2d714610225...
```

which disassembles into misaligned or invalid instructions:

```
LOG4
JUMPI
INVALID
```

```
4 INVALID
5 GT
6 PUSH2 0066
```

Such insertions corrupt instruction alignment and break control flow reconstruction, significantly hindering reverse engineering and potentially leading to undeployable bytecode.

FalseBranch injects syntactically valid but unreachable conditional branches, inflating the CFG without altering runtime behavior:

```
1 INVALID
2 PUSH0
3 INVALID
4 INVALID
5 NOT
6 PUSH10 86e5f886bac5fcadf1c6
7 SWAP11
8 INVALID
9 JUMPI
```

These dead paths force conservative exploration by static analyzers and increase symbolic execution cost.

Flower inserts syntactically valid but semantically irrelevant instructions:

```
1 PUSH2 08a6
2 JUMP
3 PUSH1 06
4 JUMPDEST
```

Although unreachable, these sequences pollute the disassembled output, inflate bytecode size, and degrade pattern-based detection.

Reorder rearranges independent instruction sequences while preserving semantics. For example:

```
1 PUSH1 40
2 MLOAD
3 PUSH2 010f
4 SWAP2
5 SWAP1
6 PUSH2 08a6
7 JUMP
```

Reordering disrupts expected structural patterns used in decompilers and ML-based disassembly, even though runtime behavior remains unchanged.

EVeilM [33] applies complementary transformations that emphasize operand and semantic distortion.

AddManip wraps `ADD` instructions with misleading stack operations. For example, the original arithmetic sequence:

```
PUSH1 20
ADD
```

is obfuscated as:

```
PUSH1 20
DUP2
PUSH1 00
SUB
SUB
SWAP1
POP
NOT
PUSH1 01
ADD
```

This preserves the final result but dramatically alters opcode distribution and stack behavior, misleading symbolic execution and n-gram-based classifiers.

FuncSigTransform reconstructs function selectors at runtime. The original dispatch:

```
PUSH4 8da5cb5b
EQ
PUSH2 0202
JUMPI
```

becomes:

```
PUSH4 247b5482
PUSH4 692a76d9
ADD
EQ
PUSH2 0202
JUMPI
```

This prevents direct signature matching and complicates decompilation.

SpamJumpDest inserts large numbers of unused `JUMPDEST` markers, inflating basic block counts and CFG size while leaving execution unchanged.

JumpTransform replaces fixed jump targets with computed ones. For example, the original jump:

```
PUSH2 0010
JUMPI
```

is transformed into:

```
PUSH2 0002
PUSH1 0e
ADD
JUMPI
```

Such transformations hinder control flow recovery and CFG-based matching.

HermHD [34] proposes additional techniques but is excluded from this study due to repeated failures on real-world contracts (see Section 8.4), frequently producing invalid or semantically inconsistent bytecode.

In summary, BOSC emphasizes structural and syntactic distortion, whereas EVeilM targets operand-level and semantic rewriting. Although conceptually compatible, combining them requires careful handling to maintain correctness. In this work, the two tools are evaluated independently.

8.4. Datasets

8.4.1. Data Collection

We rely on two real-world datasets. The first, PhishingHook [55, 56, 61], contains 7,000 EVM bytecode samples: 3,458 labeled as malicious (phishing) and 3,542 as benign. All malicious samples are used in phishing-related experiments.

The second dataset, HoneyBadger [138], consists of 546 unique honeypot contracts. To obtain a balanced evaluation set, an equal number of benign contracts are sampled from the benign portion of PhishingHook, yielding 1,092 samples for the honeypot analysis.

8.4.2. Data Preprocessing

To assess the impact of bytecode obfuscation on malware detection, we generate obfuscated variants of all samples using BOSC and EVeilM. Each tool supports four transformation techniques (see Section 8.3).

All valid non-identity combinations are systematically applied, resulting in 15 unique configurations per obfuscator. The complete transformation sets are reported in Tables 8.1 (BOSC) and 8.2 (EVeilM).

8.4.3. Performance-Relevant Dataset Metrics

To quantify the structural effects of obfuscation, we measure average bytecode size under each configuration. Bytecode length is relevant not only for storage and transfer overhead, but also because larger contracts incur higher deployment and execution gas costs (see Section 8.5.5).

Fig. 8.1 and Fig. 8.2 report average bytecode sizes produced by BOSC and EVeilM, respectively. Values are computed over all outputs, regardless of final validity; validity criteria are defined in Section 8.5.2.

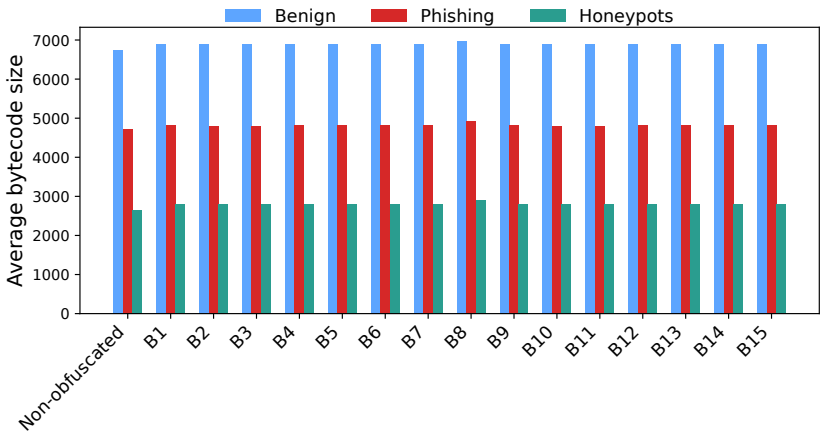

Figure 8.1.: Average size (in bytes) of contracts from the PhishingHook and HoneyBadger datasets, comparing raw and BOSC-obfuscated versions.

BOSC introduces minimal and highly consistent growth across all contract classes. Benign samples increase slightly from 6,729 bytes (baseline) to roughly 6,890–6,893 bytes (+2.3%). Phishing samples rise from 4,708 bytes to about 4,808-4,818 bytes (+2.1%). Honeypots, smaller at baseline (2,648 bytes), increase to 2,794-2,902 bytes (+5.5% to +9.6%). Overall, BOSC produces lightweight and predictable changes consistent with its transformations (*e.g.*, FalseBranch, Reorder, Flower).

In contrast, EVeilM exhibits substantially higher variability across configurations. Transformations dominated by *AddManip* (*e.g.*, E1, E5, E9, E13) produce pronounced inflation: benign samples increase to 7,259-7,838 bytes (+7.9% to +16.5%), phishing samples to 5,061-5,436 bytes (+7.5% to +15.4%), and honeypots to 2,883-3,193 bytes (+8.9% to +20.6%). These effects result from heavy stack manipulation and expanded execution paths.

Conversely, configurations dominated by *FuncSigTransform* (*e.g.*, E2, E6, E10) often produce bytecode smaller than the original, with phishing samples reduced by 18-25%, due to opcode-level re-encoding and compressed dispatch logic.

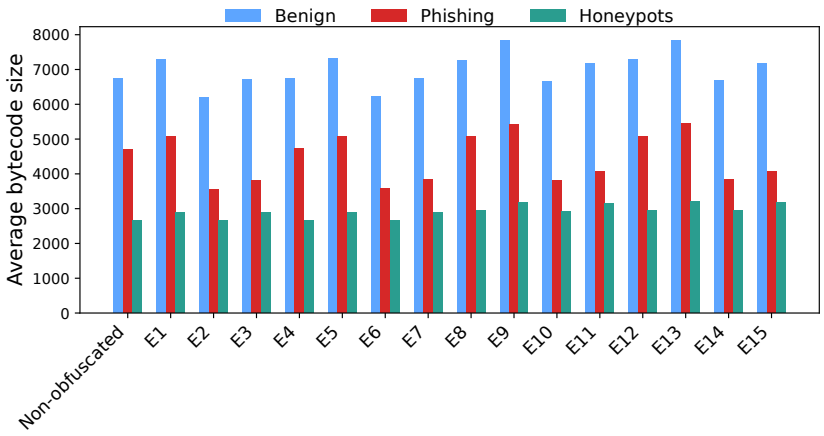

Figure 8.2.: Average size (bytes) of contracts from the PhishingHook and HoneyBadger datasets, comparing raw and EVeilM-obfuscated versions.

Taken together, EVeilM exhibits a bimodal footprint: some configurations inflate bytecode substantially, others consistently shrink it, depending on whether new logic is inserted or existing components are restructured.

To contextualize structural change beyond size, we analyze opcode-frequency distributions across all contract classes. Fig. 8.3 shows the baseline distribution dominated by stack operations (`PUSH`, `DUP`, `SWAP`), arithmetic instructions, and control-flow primitives (`JUMP`, `JUMPDEST`).

Under BOSC (Fig. 8.4), distributions remain broadly consistent, reflecting its lightweight rewriting and limited junk insertion. Under EVeilM (Fig. 8.5), opcode frequencies shift dramatically: stack-related opcodes (*e.g.*, `PUSH1`, `PUSH2`, `DUP1`, `DUP2`) increase by an order of magnitude, and control-flow instructions surge similarly. These shifts stem from aggressive path detours, opaque predicates, and extensive stack manipulation.

Overall, BOSC preserves the statistical signature of original bytecode, whereas EVeilM fundamentally reshapes it through substantial opcode inflation.

8.5. Evaluation

8.5.1. Experimental Setup

All experiments are executed across three compute nodes. The first is a Graphics Processing Unit (GPU)-enabled machine equipped with an Intel Core i7-14700KF CPU (28 cores, up to 5.5 GHz), 64 GiB of RAM, and an NVIDIA GeForce RTX 4090 GPU with 24 GiB of VRAM, running CUDA 12.8. The remaining two nodes are Central Processing Unit (CPU)-only systems, each featuring an Intel Xeon E5-2683 CPU (64 cores, 2.10 GHz) and 125 GiB of RAM.

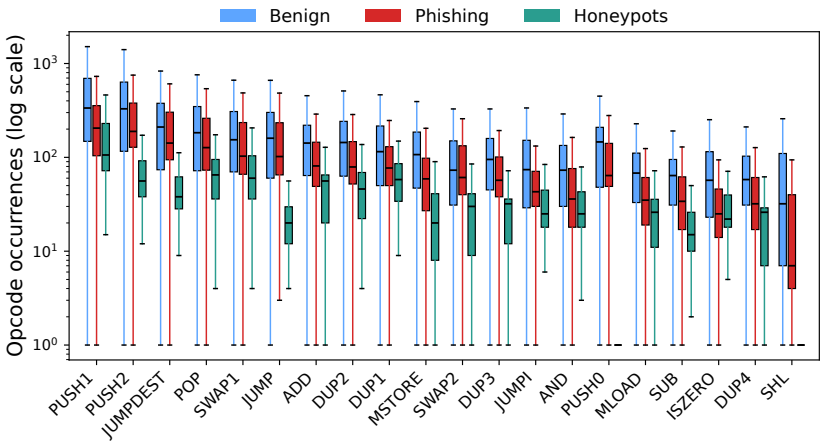

Figure 8.3.: Distribution of the 20 most frequent opcodes in non-obfuscated bytecode from the PhishingHook and HoneyBadger datasets.

Table 8.1.: Validity and GED of CFGs from BOSC-obfuscated bytecodes, for Benign (†), Phishing (‡), and Honeypot (§) samples. C: Combo, I: Incomplete, FB: FalseBranch, F: Flower, R: Reorder.

Obfuscation Techniques					CFG Metrics (Heimdall)						CFG Metrics (EVMLiSA)					
C	I	FB	F	R	Valid†	Valid‡	Valid§	GED†	GED‡	GED§	Valid†	Valid‡	Valid§	GED†	GED‡	GED§
B1	✓	✗	✗	✗	709 (20%)	750 (22%)	164 (30%)	0.91	0.91	0.90	780 (22%)	774 (22%)	166 (30%)	0.94	0.95	0.91
B2	✗	✓	✗	✗	994 (28%)	1034 (30%)	207 (38%)	0.90	0.91	0.90	1098 (31%)	1125 (33%)	210 (38%)	0.94	0.95	0.91
B3	✓	✓	✗	✗	1019 (29%)	1144 (33%)	189 (35%)	0.90	0.91	0.87	1137 (32%)	1219 (35%)	201 (37%)	0.94	0.95	0.91
B4	✗	✗	✓	✗	-	-	-	-	-	-	-	-	-	-	-	-
B5	✓	✗	✓	✗	651 (18%)	767 (22%)	161 (29%)	0.92	0.90	0.91	743 (21%)	817 (24%)	161 (29%)	0.95	0.95	0.92
B6	✗	✓	✓	✗	943 (27%)	1010 (29%)	202 (37%)	0.90	0.90	0.90	1089 (31%)	1118 (32%)	206 (38%)	0.94	0.95	0.91
B7	✓	✓	✓	✗	973 (27%)	1116 (32%)	212 (39%)	0.89	0.91	0.87	1122 (32%)	1226 (35%)	229 (42%)	0.93	0.95	0.91
B8	✗	✗	✗	✓	-	-	-	-	-	-	-	-	-	-	-	-
B9	✓	✗	✗	✓	694 (20%)	764 (22%)	160 (29%)	0.91	0.91	0.91	782 (22%)	787 (23%)	163 (30%)	0.94	0.95	0.92
B10	✗	✓	✗	✓	980 (28%)	1034 (30%)	203 (37%)	0.90	0.91	0.90	1086 (31%)	1130 (33%)	209 (38%)	0.94	0.95	0.91
B11	✓	✓	✗	✓	1039 (29%)	1114 (32%)	196 (36%)	0.90	0.91	0.88	1150 (32%)	1191 (34%)	218 (40%)	0.94	0.95	0.91
B12	✗	✗	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-
B13	✓	✗	✓	✓	664 (19%)	738 (21%)	171 (31%)	0.91	0.90	0.91	759 (21%)	791 (23%)	174 (32%)	0.94	0.95	0.91
B14	✗	✓	✓	✓	946 (27%)	1009 (29%)	202 (37%)	0.90	0.90	0.90	1089 (31%)	1128 (33%)	207 (38%)	0.94	0.95	0.91
B15	✓	✓	✓	✓	1004 (28%)	1155 (33%)	190 (35%)	0.89	0.91	0.87	1175 (33%)	1265 (37%)	207 (38%)	0.93	0.95	0.91

DL-based models are trained and evaluated using PyTorch v2.5, together with PyTorch Geometric v2.6 for graph-based architectures. Classical ML models are implemented using Scikit-learn v1.5.

8.5.2. Bytecode Validity Analysis

To ensure that obfuscated SCs remain deployable and functionally equivalent to their original versions, we perform a two-step validation process: *(i)* static analysis using Heimdall [280] and EVMLiSA [281], and *(ii)* symbolic execution using hevm [282]. Together, these steps assess both structural soundness and runtime executability, ensuring that transformed bytecode remains valid within EVM constraints.

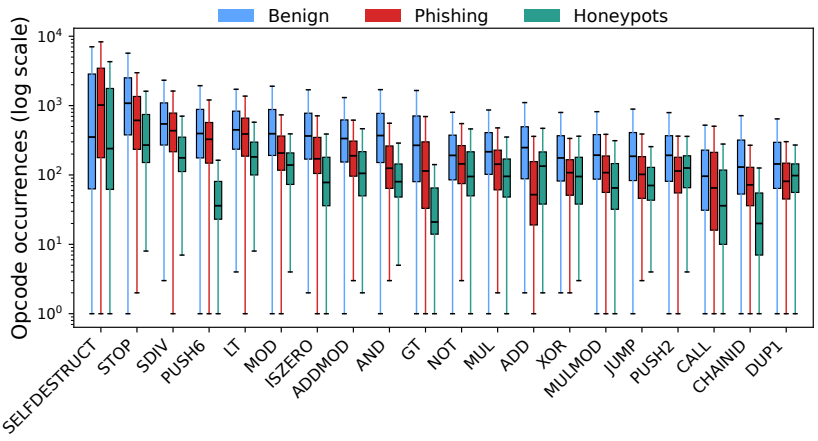

Figure 8.4.: Distribution of the 20 most frequent opcodes in BOSC-generated bytecode from the PhishingHook and HoneyBadger datasets.

We deliberately do not rely on concrete execution of the obfuscated contracts. Running a contract requires a deployed environment, specific calldata, and concrete blockchain state, and can only confirm correctness for individual execution traces. In contrast, our goal is to assess deployability and semantic preservation across several possible inputs and storage states. Static analysis and symbolic execution therefore provide a conservative, tool-supported approximation: failures in CFG reconstruction or symbolic execution indicate malformed bytecode or semantic divergence, and are treated as violations of validity or equivalence.

In the first step, Heimdall, a Rust-based EVM analysis framework, and EVMLiSA, a static analyzer based on abstract interpretation and implemented on top of LiSA [284], attempt to reconstruct the CFG for each obfuscated contract. Successful CFG extraction serves as a proxy for syntactic and structural correctness. Failures indicate malformed bytecode or invalid instruction sequences that would prevent deployment or static analysis on-chain.

In the second step, we use hevm's symbolic execution engine to verify that the obfuscated bytecode can be executed without triggering invalid opcode errors, crashes, or unresolved execution paths. This confirms that the transformed bytecode respects EVM runtime semantics and can, in principle, be deployed and interacted with.

We apply this validation pipeline to all generated samples. With BOSC, both analyzers report consistently low validity across configurations. As shown in Table 8.1, Heimdall reconstructs valid CFGs for only 18-29% of benign, 21-33% of phishing, and 29-39% of honeypot samples, while EVMLiSA reports similar ranges (21-33%, 22-36%, and 30-42%). Three configurations (B4, B8, B12) fail entirely under both tools, reflecting the structural brittleness of *Flower* and *Reorder* when applied in isolation.

The error breakdown in Table 8.3 shows that these failures are dominated by out-of-bounds instruction decoding and invalid jump targets, consistent with the effects of *Incomplete* and malformed control flow layouts.

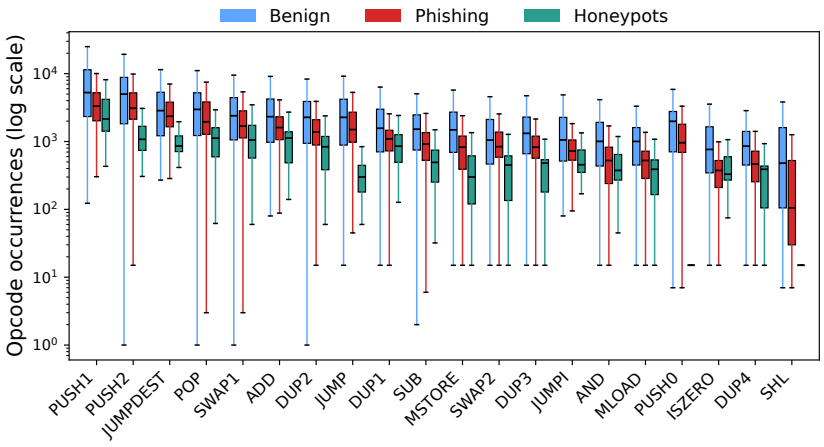

Figure 8.5.: Distribution of the 20 most frequent opcodes in EVeilM-generated bytecode from the PhishingHook and HoneyBadger datasets.

Table 8.2.: Validity and GED of CFGs from EVeilM-obfuscated bytecodes, for Benign (†), Phishing (‡), and Honeypot (§) samples. A: AddManip, F: FuncSigTransform, S: SpamJumpDest, J: JumpTransform.

Obfuscation Techniques					CFG Metrics (Heimdall)						CFG Metrics (EVMLiSA)					
C	A	F	S	J	Valid†	Valid‡	Valid§	GED†	GED‡	GED§	Valid†	Valid‡	Valid§	GED†	GED‡	GED§
E1	✓	✗	✗	✗	3478 (98%)	3442 (100%)	536 (98%)	0.75	0.84	0.82	3446 (97%)	3347 (97%)	529 (97%)	0.77	0.91	0.81
E2	✗	✓	✗	✗	3385 (96%)	3040 (88%)	542 (99%)	0.80	0.87	0.96	2335 (66%)	2583 (75%)	405 (74%)	0.67	0.84	0.75
E3	✓	✓	✗	✗	3376 (95%)	3039 (88%)	540 (99%)	0.83	0.94	0.98	3106 (88%)	2929 (85%)	475 (87%)	0.79	0.93	0.90
E4	✗	✗	✓	✗	3492 (99%)	3449 (100%)	534 (98%)	0.64	0.70	0.50	1934 (55%)	2597 (75%)	259 (47%)	0.54	0.75	0.33
E5	✓	✗	✓	✗	3468 (98%)	3419 (99%)	531 (97%)	0.75	0.84	0.82	3355 (95%)	3361 (97%)	536 (98%)	0.77	0.91	0.81
E6	✗	✓	✓	✗	3384 (96%)	3041 (88%)	541 (99%)	0.81	0.87	0.97	2029 (57%)	2314 (67%)	197 (36%)	0.67	0.83	0.73
E7	✓	✓	✓	✗	3370 (95%)	3040 (88%)	540 (99%)	0.83	0.94	0.98	3104 (88%)	2903 (84%)	475 (87%)	0.79	0.93	0.90
E8	✗	✗	✗	✓	3387 (96%)	3335 (96%)	545 (100%)	0.93	0.99	0.98	3540 (100%)	3454 (100%)	546 (100%)	0.92	0.98	0.98
E9	✓	✗	✗	✓	3386 (96%)	3334 (96%)	545 (100%)	0.93	0.99	0.98	3541 (100%)	3455 (100%)	546 (100%)	0.92	0.98	0.98
E10	✗	✓	✗	✓	3232 (91%)	2919 (84%)	542 (99%)	0.93	0.98	0.99	3348 (95%)	3020 (87%)	546 (100%)	0.91	0.98	0.97
E11	✓	✓	✗	✓	3233 (91%)	2919 (84%)	542 (99%)	0.93	0.98	0.99	3352 (95%)	3022 (87%)	546 (100%)	0.91	0.98	0.97
E12	✗	✗	✓	✓	3389 (96%)	3338 (97%)	544 (100%)	0.93	0.99	0.98	540 (100%)	3453 (100%)	546 (100%)	0.92	0.98	0.98
E13	✓	✗	✓	✓	3386 (96%)	3337 (97%)	545 (100%)	0.93	0.99	0.98	3541 (100%)	3455 (100%)	546 (100%)	0.92	0.98	0.98
E14	✗	✓	✓	✓	3231 (91%)	2920 (84%)	541 (99%)	0.93	0.98	0.99	3344 (94%)	3017 (87%)	546 (100%)	0.91	0.98	0.97
E15	✓	✓	✓	✓	3231 (91%)	2920 (84%)	542 (99%)	0.93	0.98	0.99	3347 (94%)	3014 (87%)	546 (100%)	0.91	0.98	0.97

Configurations combining *Incomplete* with *FalseBranch* (B3, B7, B11, B15) yield the highest BOSC validity, reaching the upper end of its range (*e.g.*, 28-33% under Heimdall and 32-36% under EVMLiSA; see Table 8.1). This suggests that the structurally well-formed dead branches introduced by *FalseBranch* partially offset the malformed instruction sequences generated by *Incomplete*, restoring enough syntactic structure for CFG reconstruction and symbolic execution.

EVeilM exhibits a markedly different profile. Across all fifteen configurations, outputs remain consistently valid: Heimdall validates 91-99% of benign samples, 84-99% of phishing samples, and 97-100% of honeypots (see Table 8.2). EVMLiSA confirms these results, reaching up to 100% validity across all classes. No EVeilM configuration fails entirely, and the residual errors reported in Table 8.4 are attributable to analysis limitations, such as symbolic execution timeouts (Heimdall) or abstract-domain singleton exceptions (EVMLiSA), rather than malformed bytecode.

Table 8.3.: Error analysis of Heimdall and EVMLiSA-generated CFGs using BOSC configurations. Each cell reports results for Benign / Phishing / Honeypot samples. C: Combo.

	Heimdall				EVMLiSA	
C	Out-of-Bounds	Symbolic Ex. Fail.	Failed First Op.	Invalid Target	Out-of-Bounds	Other Errors
B1	2772 / 2605 / 376	0 / 0 / 0	0 / 3 / 0	56 / 96 / 6	2760 / 2675 / 380	2 / 9 / 0
B2	2480 / 2177 / 331	0 / 1 / 0	0 / 33 / 0	56 / 179 / 6	2443 / 2321 / 336	1 / 12 / 0
B3	2461 / 2077 / 339	0 / 0 / 0	0 / 18 / 0	56 / 179 / 6	2401 / 2225 / 345	4 / 14 / 0
B4	3486 / 3362 / 540	0 / 0 / 0	0 / 0 / 0	56 / 96 / 6	3542 / 3458 / 546	0 / 0 / 0
B5	2828 / 2589 / 379	1 / 0 / 0	0 / 0 / 0	56 / 96 / 6	2796 / 2637 / 385	3 / 4 / 0
B6	2532 / 2199 / 336	0 / 1 / 0	0 / 33 / 0	56 / 179 / 6	2451 / 2328 / 340	2 / 12 / 0
B7	2503 / 2100 / 314	0 / 0 / 0	0 / 17 / 0	56 / 179 / 6	2417 / 2228 / 317	3 / 4 / 0
B8	3542 / 3458 / 546	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	3542 / 3458 / 546	0 / 0 / 0
B9	2782 / 2588 / 380	0 / 0 / 0	0 / 4 / 0	56 / 96 / 6	2760 / 2666 / 383	0 / 5 / 0
B10	2492 / 2177 / 335	0 / 1 / 0	0 / 33 / 0	56 / 179 / 6	2454 / 2321 / 337	2 / 7 / 0
B11	2437 / 2115 / 328	0 / 0 / 0	0 / 17 / 0	56 / 179 / 6	2388 / 2262 / 328	4 / 5 / 0
B12	3486 / 3362 / 540	0 / 0 / 0	0 / 0 / 0	56 / 96 / 6	3542 / 3458 / 546	0 / 0 / 0
B13	2816 / 2618 / 369	0 / 0 / 0	0 / 0 / 0	56 / 96 / 6	2779 / 2661 / 372	4 / 6 / 0
B14	2527 / 2201 / 336	0 / 1 / 0	0 / 33 / 0	56 / 179 / 6	2452 / 2324 / 339	1 / 6 / 0
B15	2476 / 2054 / 337	0 / 1 / 0	0 / 20 / 0	56 / 179 / 6	2364 / 2187 / 339	3 / 6 / 0

Overall, BOSC's structural perturbations frequently compromise syntactic well-formedness, while EVeilM preserves bytecode validity across all configurations.

8.5.3. Control Flow Divergence Analysis

A CFG captures the execution structure of a SC, where nodes represent basic blocks and edges denote possible control transfers between them. This abstraction reflects runtime behavior and provides the foundation for graph-based detection approaches, such as Graph Neural Networks (GNNs), that rely on structural properties to infer program semantics.

For BOSC, all variants (B1-B15) result in CFGs within a narrow size range: Heimdall extracts 17-20 nodes and 16-18 edges, while EVMLiSA reports slightly larger but equally constrained values. For EVeilM, configurations E1, E4, and E5 generate larger CFGs than BOSC, whereas E8-E15 consistently collapse to minimal structures (2 nodes and 1 edge under Heimdall; 2-3 nodes and 1-2 edges under EVMLiSA).

To quantify structural divergence introduced by obfuscation, we compute the Graph Edit Distance (GED) [285] between raw and obfuscated CFGs. GED measures the minimum cost of transforming one graph into another through insertions, deletions, or substitutions. We approximate this cost using the Hungarian algorithm [286], following prior work on CFG analysis [287]. Node substitutions incur a cost equal to half the absolute difference in node degrees, while node and edge insertions or deletions cost 1.0. All values are normalized to $[0, 1]$, where higher scores indicate greater structural divergence and, by extension, stronger obfuscation.

As shown in Table 8.1, BOSC produces consistently high and tightly clustered GED values under both extractors. With Heimdall, scores range from 0.87-0.92 across all contract classes, while EVMLiSA reports slightly higher values (0.91-0.95), confirming that BOSC induces uniform structural distortion.

Table 8.4.: Error analysis of Heimdall and EVMLiSA-generated CFGs using EVeilM configurations. Each cell reports results for Benign / Phishing / Honeypot samples. C: Combo.

	Heimdall					EVMLiSA		
C	Out-of-Bounds	Symbolic Ex. Fail.	Failed First Op.	Invalid Target	Other Errors	Out-of-Bounds	Singleton Ex.	Other Errors
E1	52 / 16 / 10	5 / 0 / 0	2 / 0 / 0	0 / 0 / 0	5 / 0 / 0	0 / 0 / 0	94 / 111 / 17	2 / 0 / 0
E2	2 / 1 / 4	2 / 1 / 4	0 / 0 / 0	152 / 417 / 0	1 / 0 / 0	152 / 417 / 0	1048 / 455 / 141	7 / 3 / 0
E3	9 / 2 / 2	4 / 0 / 4	1 / 0 / 0	152 / 417 / 0	0 / 0 / 0	152 / 417 / 0	282 / 110 / 70	2 / 2 / 1
E4	36 / 9 / 12	1 / 0 / 0	6 / 0 / 0	0 / 0 / 0	7 / 0 / 0	0 / 0 / 0	1606 / 859 / 287	2 / 2 / 0
E5	63 / 34 / 13	3 / 0 / 2	4 / 2 / 0	0 / 0 / 0	4 / 3 / 0	0 / 0 / 0	182 / 91 / 9	5 / 6 / 1
E6	3 / 0 / 1	3 / 0 / 4	0 / 0 / 0	152 / 417 / 0	0 / 0 / 0	152 / 417 / 0	1353 / 722 / 346	8 / 5 / 3
E7	11 / 1 / 2	9 / 0 / 4	0 / 0 / 0	152 / 417 / 0	0 / 0 / 0	152 / 417 / 0	282 / 136 / 71	4 / 2 / 0
E8	0 / 0 / 0	155 / 123 / 1	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	2 / 4 / 0
E9	0 / 1 / 0	156 / 123 / 1	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	1 / 3 / 0
E10	0 / 0 / 0	158 / 122 / 4	0 / 0 / 0	152 / 417 / 0	0 / 0 / 0	152 / 417 / 0	41 / 15 / 0	1 / 6 / 0
E11	0 / 0 / 0	157 / 122 / 4	0 / 0 / 0	152 / 417 / 0	0 / 0 / 0	152 / 417 / 0	37 / 14 / 0	1 / 5 / 0
E12	0 / 0 / 0	153 / 120 / 2	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	2 / 5 / 0
E13	0 / 0 / 0	156 / 121 / 1	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	0 / 0 / 0	1 / 3 / 0
E14	0 / 0 / 0	159 / 121 / 5	0 / 0 / 0	152 / 417 / 0	0 / 0 / 0	152 / 417 / 0	45 / 19 / 0	1 / 5 / 0
E15	0 / 0 / 0	159 / 121 / 4	0 / 0 / 0	152 / 417 / 0	0 / 0 / 0	152 / 417 / 0	41 / 23 / 0	2 / 4 / 0

EVeilM exhibits a substantially wider spectrum of structural impact (Table 8.2). Under Heimdall, early configurations (E1-E7) produce moderate divergence: 0.64-0.83 for benign samples, 0.70-0.94 for phishing, and 0.50-0.98 for honeypots, whereas later variants (E8-E15) converge to near-maximal disruption with GED values of 0.93-0.99 across all classes. EVMLiSA reflects the same pattern with lower sensitivity on E1-E7 (0.33-0.93) and highly stable, near-maximal scores for E8-E15 (0.91-0.98).

Overall, BOSC induces consistently high CFG divergence through compact rewrites, whereas EVeilM spans a broader gradient of structural effects, from moderate perturbations in E1-E7 to extreme CFG collapse in E8-E15.

8.5.4. Semantic Equivalence Analysis

To assess whether obfuscation preserves the malicious functionality of SCs, we evaluate the semantic equivalence between each obfuscated malware sample and its original version. We perform this analysis using hevm [282], a symbolic execution engine designed for formal analysis of EVM bytecode. This enables us to determine whether two contracts behave identically for all possible inputs and initial storage states, without requiring concrete execution traces.

In this setting, equivalence is defined in terms of observable behavior: two contracts are considered semantically equivalent if they produce identical return values, leave persistent storage in the same final state, and either both succeed or both fail during execution. Storage refers to the contract's persistent key-value memory, modified via `SSTORE` and accessed via `SLOAD`, while calldata constitutes the transient, read-only input region containing function selectors and parameters for external calls. Gas usage, event emissions, and internal control flow ordering are not considered; only externally visible functional behavior is compared.

This equivalence procedure is applied to all valid obfuscated malware samples.

The results, summarized in Table 8.5, show a stark contrast between BOSC and EVeilM in their ability to preserve malicious semantics.

Table 8.5.: Semantic equivalence analysis using hevm for BOSC and EVeilM configurations. Each cell reports results for Phishing / Honeypot samples. Columns marked ∗ indicate the number of semantically equivalent malware bytecodes (% of total). Columns marked † denote failures or unresolved cases. C: Combo.

BOSC						EVeilM					
C	Equiv.*	Timeouts†	Invalid†	Unimpl.†	Killed†	C	Equiv.*	Timeouts†	Invalid†	Unimpl.†	Killed†
B1	7 (4%) / 0 (0%)	599 / 0	2675 / 380	2 / 18	0 / 10	E1	271 (18%) / 51 (11%)	1934 / 0	0 / 0	27 / 63	1 / 28
B2	25 (7%) / 0 (0%)	778 / 0	2321 / 336	4 / 24	0 / 9	E2	264 (19%) / 0 (0%)	1647 / 0	417 / 0	19 / 75	1 / 23
B3	25 (6%) / 0 (0%)	824 / 0	2225 / 345	5 / 29	0 / 7	E3	263 (18%) / 0 (0%)	1570 / 0	417 / 0	22 / 77	1 / 26
B4	0 (0%) / 0 (0%)	0 / 0	3458 / 546	0 / 0	0 / 0	E4	366 (25%) / 139 (28%)	1956 / 0	0 / 0	19 / 27	4 / 31
B5	9 (4%) / 0 (0%)	607 / 0	2637 / 385	2 / 16	0 / 8	E5	274 (18%) / 52 (11%)	1942 / 0	0 / 0	25 / 63	4 / 24
B6	25 (7%) / 0 (0%)	771 / 0	2328 / 340	4 / 24	0 / 10	E6	263 (18%) / 0 (0%)	1591 / 0	417 / 0	20 / 73	2 / 22
B7	27 (7%) / 0 (0%)	819 / 0	2228 / 317	5 / 30	0 / 11	E7	263 (18%) / 0 (0%)	1580 / 0	417 / 0	23 / 81	3 / 18
B8	0 (0%) / 0 (0%)	0 / 0	3458 / 546	0 / 0	0 / 0	E8	262 (16%) / 0 (0%)	1755 / 0	0 / 0	22 / 76	1 / 19
B9	9 (5%) / 0 (0%)	592 / 0	2666 / 383	1 / 12	0 / 7	E9	262 (16%) / 0 (0%)	1769 / 0	0 / 0	22 / 78	1 / 18
B10	25 (7%) / 0 (0%)	779 / 0	2321 / 337	4 / 24	0 / 7	E10	262 (17%) / 0 (0%)	1466 / 0	417 / 0	18 / 83	0 / 15
B11	22 (6%) / 0 (0%)	808 / 0	2262 / 328	6 / 31	1 / 8	E11	262 (17%) / 0 (0%)	1477 / 0	417 / 0	18 / 85	0 / 16
B12	0 (0%) / 0 (0%)	0 / 0	3458 / 546	0 / 0	0 / 0	E12	262 (16%) / 0 (0%)	1779 / 0	0 / 0	23 / 75	1 / 19
B13	7 (3%) / 0 (0%)	581 / 0	2661 / 372	6 / 16	0 / 9	E13	262 (16%) / 0 (0%)	1780 / 0	0 / 0	23 / 79	2 / 16
B14	25 (7%) / 0 (0%)	771 / 0	2324 / 339	4 / 25	0 / 8	E14	262 (17%) / 0 (0%)	1465 / 0	417 / 0	19 / 84	1 / 18
B15	24 (6%) / 0 (0%)	854 / 0	2187 / 339	7 / 36	0 / 12	E15	262 (17%) / 0 (0%)	1464 / 0	417 / 0	18 / 85	0 / 24

Across all BOSC configurations, equivalence rates are extremely low: phishing samples consistently fall between 0-7% equivalent, and honeypots exhibit 0% equivalence across the board. The dominant source of inequivalence aligns with the syntactic failures highlighted earlier: large volumes of *Invalid Bytecode* (*e.g.*, 2,187-3,458 invalid phishing samples and 317-546 invalid honeypots per configuration), typically caused by *Incomplete* and *Flower*. Malformed instruction sequences prevent hevm from symbolically executing the contract, immediately breaking semantic equivalence. This indicates that BOSC frequently produces obfuscated bytecode that is not only difficult to analyze, but also unsuitable for semantics-preserving transformation, undermining its practical use for realistic malware obfuscation. Additional inequivalence arises from *Execution Timeouts*, commonly 581-854 per phishing configuration, due to the 60 s timeout imposed to prevent Bitwuzla [288], the SMT solver used by hevm, from exploring excessively large symbolic paths. A small number of *Unimplemented* opcodes and occasional *Killed Processes* contribute minor additional failures.

EVeilM exhibits a markedly different profile. In configurations E1-E7, semantics are preserved for a substantially larger share of phishing samples (15-19%) and a nontrivial portion of honeypots (up to 28% for E4 and 11% for E1 and E5). These configurations apply transformations such as *AddManip* and *FuncSigTransform*, which modify bytecode layout while largely retaining the original control logic. The primary sources of inequivalence are *Execution Timeouts* (1,464-1,956 per phishing configuration) and a moderate number of *Unimplemented* cases, stemming from newly introduced opcode patterns that hevm cannot symbolically process.

By contrast, configurations E8-E15 apply more aggressive control flow rewriting (*SpamJumpDest*, *JumpTransform*) that collapses CFG structure. Despite this, they still achieve 15-17% equivalence on phishing samples, though honeypot equivalence drops to 0%. Failure modes shift accordingly: invalid bytecode is far less frequent (0-417 cases), timeouts remain high (1,464-1,779 for phishing), and *Unimplemented* paths increase, reaching up to 85 cases in E10-E15 for honeypots, indicating that these transformations introduce symbolic execution paths involving opcodes or stack behaviors currently unsupported by hevm.

8.5.5. Gas Cost Analysis

We analyse the gas cost of obfuscation at two levels. First, the full-bytecode gas cost is computed by summing the minimum gas required for each opcode in the deployed (runtime) bytecode, following the EVM specifications. Although this does not reflect the actual deployment gas, it provides an upper-bound estimate of the maximum execution cost inherent to the deployed bytecode.

Second, to approximate runtime gas statically, we compute a CFG-based gas cost by summing the base cost of all opcodes contained in reachable basic blocks within the CFGs reconstructed by Heimdall and EVMLiSA. This excludes unreachable junk code introduced by obfuscators and therefore more closely reflects the gas that can actually be consumed during execution.

The results in Table 8.6 reveal a sharp contrast between BOSC and EVeilM.

BOSC induces extreme full-bytecode inflation. Deployment gas increases by approximately +7,000% to +12,000% for benign and phishing samples, and up to +9,939.67% for honeypots (B15). These increases stem from BOSC's heavy insertion of opaque or unreachable instruction sequences (*e.g.*, *Incomplete*, *Flower*, *Reorder*), all of which are counted during deployment even if never executed.

In contrast, BOSC's CFG-based gas cost collapses. Heimdall reports reductions of roughly −87% to −96%, as only a few reachable blocks remain after obfuscation. EVMLiSA shows smaller but still substantial reductions (−56% to −80%), reflecting its more permissive CFG extraction. Thus, BOSC produces bytecode that is extremely costly to deploy but contains very little reachable logic, highlighting the severe structural disruption introduced by its transformations.

EVeilM exhibits a fundamentally different pattern. Full-bytecode gas remains close to baseline, ranging from approximately −4% to +7% across all classes, with honeypots peaking at only +6.42% in E9. This indicates that EVeilM preserves overall bytecode size far more aggressively than BOSC.

However, EVeilM's CFG-based gas cost sharply collapses for configurations E8-E15. Under Heimdall, these variants show near-total reductions of −99.4% to −99.5%, while EVMLiSA reports reductions between −90% and −98.8%. These values align with the extreme CFG collapse observed earlier: for E8-E15, almost all executable logic disappears, leaving only minimal entry/exit structure.

Overall, BOSC inflates deployment gas dramatically while eliminating most reachable logic, whereas EVeilM preserves realistic deployment size despite aggressively collapsing the executable control flow. These contrasting behaviours illustrate how structural versus semantic obfuscation strategies impose fundamentally different gas footprints on EVM bytecode.

8.5.6. Detection of Obfuscated Malware

Methodology, Training, and Validation. To evaluate the robustness of ML models against obfuscated malicious SCs, we assess the top-performing classifiers from PhishingHook and extend the study with a suite of GNNs. Specifically, our evaluation includes: *(i)* RF (HSC), *(ii)*

Table 8.6.: Percentage change in minimum gas cost relative to the non-obfuscated bytecode baseline, for the full bytecode and for the CFGs generated by EVMLiSA and Heimdall. Each cell reports Benign / Phishing / Honeypot results, aggregated over all validly generated bytecodes in each combo. C: Combo.

BOSC				EVeilM			
C	Δ (%) Full Bytecode	Δ (%) CFG Heimdall	Δ (%) CFG EVMLiSA	C	Δ (%) Full Bytecode	Δ (%) CFG Heimdall	Δ (%) CFG EVMLiSA
B1	+9810% / +11815% / +7398%	-93% / -94% / -90%	-64% / -79% / -63%	E1	+4% / +3% / +3%	-91% / -91% / -89%	-81% / -81% / -75%
B2	+9479% / +80560% / +9320%	-95% / -95% / -91%	-54% / -78% / -69%	E2	-4% / -4% / +0%	-98% / -98% / -91%	-85% / -85% / -36%
B3	+10029% / +9408% / +9080%	-95% / -94% / -88%	-56% / -79% / -72%	E3	+0% / -2% / +3%	-99% / -99% / -98%	-90% / -89% / -76%
B4	- / - / -	- / - / -	- / - / -	E4	+0% / +0% / +0%	-83% / -83% / -70%	-84% / -81% / -63%
B5	+9747% / +11925% / +6211%	-95% / -94% / -92%	-31% / -61% / -75%	E5	+4% / +3% / +3%	-91% / -90% / -88%	-82% / -81% / -73%
B6	+9436% / +7118% / +8673%	-93% / -95% / -91%	-59% / -74% / -71%	E6	-4% / -4% / +0%	-99% / -99% / -95%	-92% / -93% / -70%
B7	+9740% / +8538% / +8922%	-94% / -95% / -90%	-58% / -73% / -64%	E7	+4% / -2% / +3%	-99% / -99% / -98%	-90% / -90% / -77%
B8	- / - / -	- / - / -	- / - / -	E8	+3% / +2% / +3%	-99% / -99% / -99%	-98% / -99% / -96%
B9	+9500% / +11622% / +6971%	-94% / -94% / -91%	-60% / -75% / -67%	E9	+7% / +5% / +6%	-99% / -99% / -99%	-98% / -99% / -96%
B10	+9534% / +7144% / +8108%	-95% / -95% / -91%	-55% / -78% / -69%	E10	-1% / -2% / +3%	-99% / -99% / -100%	-98% / -98% / -90%
B11	+9771% / +9361% / +9429%	-95% / -95% / -88%	-58% / -80% / -65%	E11	+3% / +0% / +0%	-99% / -99% / -100%	-98% / -98% / -90%
B12	- / - / -	- / - / -	- / - / -	E12	+3% / +2% / +3%	-99% / -99% / -99%	-98% / -99% / -96%
B13	+9618% / +10043% / +6248%	-95% / -96% / -90%	-30% / -64% / -75%	E13	+7% / +5% / +4%	-99% / -99% / -99%	-98% / -99% / -96%
B14	+9491% / +7113% / +8087%	-94% / -95% / -91%	-63% / -79% / -73%	E14	-1% / -2% / +3%	-99% / -99% / -100%	-98% / -98% / -91%
B15	+9600% / +8590% / +9940%	-94% / -95% / -87%	-57% / -78% / -63%	E15	+3% / +0% / +0%	-99% / -99% / -100%	-98% / -98% / -90%

ECA+EfficientNet (CVM), *(iii)* SCSGuard (LLM), and five GNN architectures: Graph Attention Network (GAT), Graph Convolutional Network (GCN), Graph Isomorphism Network (GIN), Graph Sample and Aggregate (GraphSAGE), and Topology Adaptive Graph Convolutional Network (TAG). For convenience, we refer to the detectors in *(i)-(iii)* as the PhishingHook classifiers.

The RF takes opcode-frequency vectors as input. ECA+EfficientNet extends R2D2 [254] by mapping bytecode to RGB images using hexadecimal color encoding, which are then processed by EfficientNet [100] enhanced with ECA channel attention [101]. SCSGuard models bytecode as sequences of 6-grams, tokenized, integer-encoded, and padded prior to entering a sequence model. GNNs operate on CFGs extracted via Heimdall and EVMLiSA; nodes represent basic blocks, edges represent control transfers, and node features are initialized with CodeBERT embeddings [252].

Hyperparameters are optimized using Optuna [259] with 10-fold cross-validation. Final models are retrained with optimal hyperparameters on the full raw training set and then evaluated independently on each of the 12 BOSC and 15 EVeilM configurations.

To ensure fair comparison, all classifiers, including histogram, vision, language, and graph-based models, are evaluated on identical subsets. Because GNNs require valid CFGs, we restrict every model to the samples for which Heimdall or EVMLiSA successfully extract a CFG, reporting results separately for both extractors.

For honeypots, we follow the protocol in section 8.4.1: 546 malicious samples and 546 benign samples (uniformly sampled). The same split is used across all models and obfuscation settings.

Detection Performance under Obfuscation. Phishing detection (Fig. 8.6) proves substantially more challenging than honeypot detection: all models exhibit lower and more variable F1 scores on phishing samples.

GNNs are the most heavily affected. Under Heimdall, several GNNs collapse to near-zero F1 for many EVeilM configurations. GAT is particularly unstable, with F1 scores approaching zero for E1, E3, E5, and E7-E15. EVMLiSA does not improve this trend: all GNNs fall below F1 < 0.20

on E8-E15. Occasional peaks exist (*e.g.*, GCN and GIN on E4), but these are isolated and surrounded by regions of near-zero performance.

RF exhibits inconsistent behavior. Under Heimdall, F1 fluctuates in a low band, with pronounced drops for BOSC. Under EVMLiSA, it occasionally spikes, but collapses sharply for E8-E9 and E12-E13.

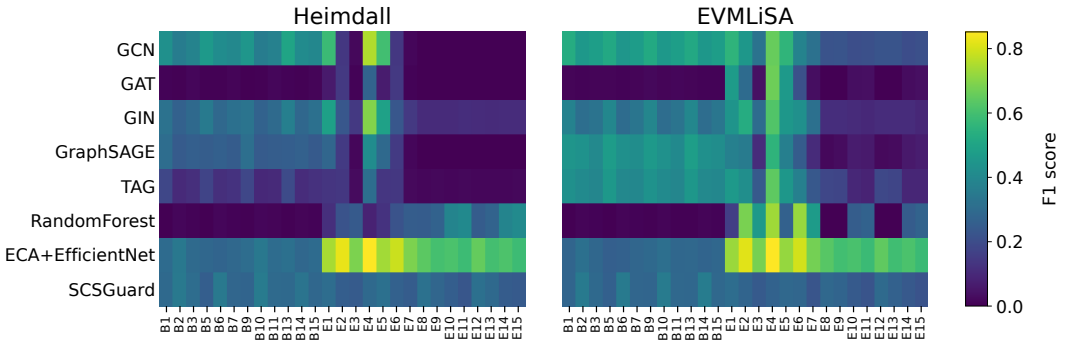

Figure 8.6.: Detection performance (F1 score) for phishing samples. Results shown for CFGs extracted via Heimdall (left) and EVMLiSA (right).

SCSGuard is more stable: it avoids total collapse but rarely achieves high F1, instead maintaining moderate and relatively uniform performance across configurations and extractors.

In contrast, ECA+EfficientNet shows the strongest robustness. Across both Heimdall and EVMLiSA, it maintains F1 $\approx$ 0.60–0.85 for E1-E7 and exhibits notably strong results on configurations such as E2 and E4. In the heatmaps, the ECA+EfficientNet row is consistently the brightest, demonstrating resistance to structural distortions introduced by EVeilM.

Configuration E4 is particularly distinctive: ECA+EfficientNet performs strongly under both extractors, whereas other models show instability or low F1, making E4 a prime candidate for interpretability analysis.

Honeypot detection (Fig. 8.7) behaves differently: all models, including GNNs, achieve consistently high F1 (often > 0.80) across all configurations and extractors. Thus, phishing serves as the more informative benchmark for evaluating robustness under obfuscation.

Overall, EVeilM induces substantial variation across classifiers: GNNs are highly fragile, RFs and SCSGuard moderately stable but inconsistent, and ECA+EfficientNet clearly the most resilient across obfuscation strategies and CFG extraction pipelines.

Interpreting ECA+EfficientNet Results via Class Activation Mapping. To understand why ECA+EfficientNet performs strongly under EVeilM, we examine the model's spatial attention using Class Activation Mapping (CAM) [289]. We analyze four phishing samples correctly classified with high confidence under configuration E4 (IDs 451, 640, 3057, 3384).

All bytecode images share a similar structure: a narrow non-padded region at the top and large black (zero-padded) areas below. Although this visual encoding limits interpretability, CAM reveals a consistent focus pattern: all models concentrate activation in the top-left portion of

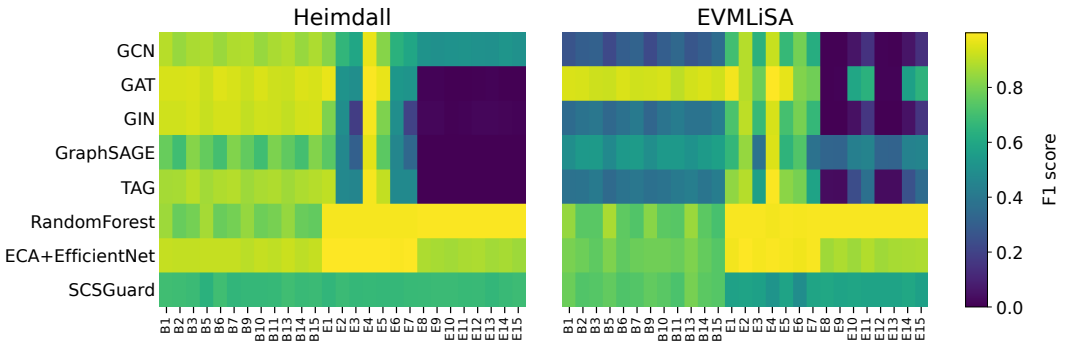

Figure 8.7.: Detection performance (F1 score) for honeypot samples. Results shown for CFGs extracted via Heimdall (left) and EVMLiSA (right).

the meaningful region. This suggests reliance on local statistical patterns, likely early byte-opcode distributions, rather than global structural features.

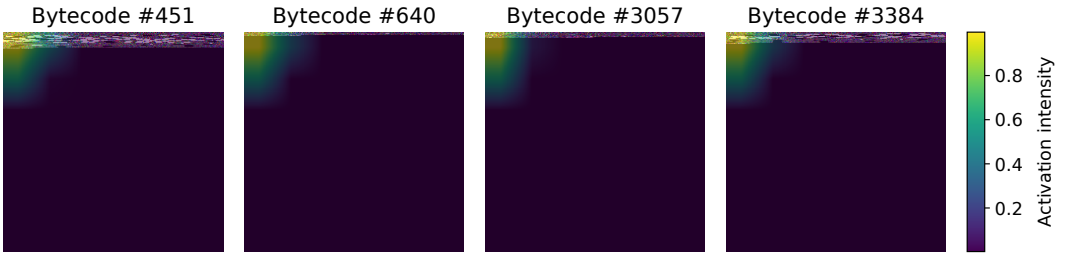

Figure 8.8.: Class Activation Mapping for RGB images derived from bytecodes 451, 640, 3057, and 3384 under EVeilM configuration E4.

Quantitatively, CAM activation metrics show: cam_max $\in [0.99994, 0.99995]$, cam_mean $\in [0.0351, 0.0423]$, cam_std $\in [0.1632, 0.1734]$. These values indicate extremely sharp, localized attention and explain the model's robustness: ECA+EfficientNet depends on compact, distortion-resistant byte patterns rather than full control flow structure, unlike GNNs or n-gram models.

Detection of Semantically Equivalent Malware. We further analyze detection performance on the subset of malware that remains semantically equivalent after obfuscation (Section 8.5.4).

For BOSC, the preserved-malware subset is extremely small (0-25 samples per configuration and extractor). Even within this limited set, trends are clear: ECA+EfficientNet detects $\approx$ 65-100% of preserved phishing samples, RF detects $\approx$ 11-28%, SCSGuard fluctuates, and GNNs detect virtually none, with only rare exceptions.

EVeilM exhibits a different profile. For semantically preserved malware, GNNs perform best: near-perfect honeypot detection, and up to 86% phishing detection for early configurations (E1-E6).

However, for E7-E15, GNN performance collapses to zero, even though these configurations preserve substantial functional equivalence for phishing samples. This collapse mirrors the patterns observed in the full-dataset heatmaps and occurs consistently across both extractors.

8.6. Lessons Learned

Our evaluation shows that the practical effectiveness of EVM bytecode obfuscation hinges on its ability to preserve both syntactic correctness and runtime soundness. As demonstrated in Section 8.5.2, BOSC frequently violates these requirements: only 18-39% of its outputs pass structural checks under Heimdall or EVMLiSA, and several configurations (B4, B8, B12) fail entirely. These failures are dominated by out-of-bounds instruction decoding and invalid jump targets, reflecting brittle control flow rewrites and malformed instruction sequences. In contrast, EVeilM consistently maintains 84-100% validity, and its remaining analysis errors stem from tool limitations rather than malformed bytecode. Overall, these results reinforce that any practical obfuscator must prioritize structural and semantic well-formedness at the EVM level.

> Obfuscators must guarantee runtime validity. EVeilM consistently produces deployable bytecode, whereas BOSC violates structural and execution constraints of the EVM.

As shown in Section 8.5.3 and Section 8.5.4, both obfuscators introduce substantial control flow distortion, but often at the cost of functional correctness. BOSC preserves malicious behavior in only 0-7% of phishing samples and in 0% of honeypots, primarily due to malformed bytecode and distorted control flow structures arising from transformations such as *Incomplete* and *Flower*. EVeilM performs better, retaining 15-19% equivalence for phishing and up to 28% for honeypots, yet equivalence remains low overall. This highlights a fundamental limitation: current bytecode-level transformations frequently couple structural perturbations with semantic drift, making them unsuitable for security-sensitive contexts without substantially stronger correctness guarantees.

> Preserving functional behavior remains difficult. Current obfuscation techniques frequently disrupt semantics, and even state-of-the-art tools maintain equivalence only in a limited subset of cases.

Regarding malware detection, the vision-based ECA+EfficientNet model emerges as the most resilient classifier in our study. Across both Heimdall and EVMLiSA extractions, it consistently maintains high phishing detection F1 scores, even in configurations where other models, including GNNs, RFs, and LLMs, collapse. As shown by the CAM analysis in Section 8.5.6, the model relies on localized byte-distribution patterns rather than explicit control flow structure. This design makes it inherently more tolerant to CFG collapse and the large-scale structural rewriting introduced by EVeilM.

> Vision-based detection is the most robust and obfuscation-tolerant approach. By exploiting local visual patterns instead of structural features, it maintains strong performance even under aggressive rewriting.

Finally, the gas cost analysis in Section 8.5.5 highlights important operational implications. BOSC inflates deployment-size gas by 7,000-12,000%, driven by extensive insertion of unreachable code, while simultaneously reducing reachable logic by up to 96% in CFG-based estimates. EVeilM exhibits a different profile: deployment gas remains near baseline (–4% to +7%), yet reachable gas collapses almost completely in configurations E8-E15. These patterns show that gas cost dynamics not only quantify overhead but also serve as an indicator of how aggressively a tool rewrites program structure.

> Gas cost behavior exposes distinct transformation trade-offs. BOSC introduces substantial deployment-gas inflation while leaving little reachable logic, whereas EVeilM preserves realistic deployment size but sharply reduces reachable logic in its more aggressive configurations.

Taken together, our findings are not intended to discredit existing obfuscation tools, but to empirically characterize their behavior and limitations when applied to real-world EVM malware. By systematically evaluating validity, semantic preservation, gas impact, and detection robustness, we show that current obfuscation strategies face inherent trade-offs between structural distortion and functional fidelity. At the same time, our results demonstrate that robust malware detection under severe structural rewriting is achievable when representations are less sensitive to control flow distortion. In this sense, the chapter provides both a diagnostic assessment of the current obfuscation landscape and constructive guidance for the design of future, semantics-aware obfuscators and obfuscation-resilient detection models.

8.7. Summary and Next Steps

This chapter presents the first comprehensive evaluation of EVM bytecode obfuscation and its implications for SC malware detection. Our findings show that deployability is a fundamental requirement that current tools satisfy unevenly. BOSC produces structurally valid bytecode in only 18-39% of cases and fails entirely under several configurations, while EVeilM consistently maintains 84-100% validity. These results highlight the need for obfuscation techniques that reliably preserve syntactic well-formedness and runtime executability.

Our analysis further indicates that current bytecode-level transformations frequently alter contract semantics. Although both BOSC and EVeilM achieve substantial control flow distortion, these structural rewrites often come at the cost of functional correctness. BOSC preserves malicious behavior in only 0-7% of phishing samples and in none of the honeypots, while EVeilM retains semantics for only 15-19% of phishing and up to 28% of honeypots in its moderate configurations. This demonstrates that existing EVM obfuscators commonly entangle structural

perturbation with behavioral drift, underscoring the need for techniques that better balance obfuscation strength and semantic fidelity.

From a programming-semantics perspective, malicious and legitimate SCs are often indistinguishable at the level of low-level correctness: both may be syntactically valid, terminate normally, and satisfy the EVM execution model. What differentiates malware is not semantic well-formedness per se, but behavioral intent: phishing contracts implement logic that intentionally misleads users, diverts assets, or violates expected economic assumptions, despite remaining functionally consistent with the language semantics. In this study, malicious behavior is therefore defined operationally, based on externally observable effects and ground-truth labels, rather than on intrinsic semantic violations. A deeper, intent-aware semantic characterization of malicious versus legitimate contracts remains an important direction for future work.

On the detection side, our evaluation shows that classifiers relying on opcode distributions, n-gram sequences, or CFGs are highly sensitive to obfuscation: their recall drops sharply once control flow is flattened, inflated with unreachable blocks, or otherwise distorted. In contrast, the vision-based ECA+EfficientNet model exhibits markedly stronger robustness. By exploiting localized visual patterns rather than explicit structural signals, it maintains predictive performance even when both sequence- and graph-based representations deteriorate under aggressive rewriting.

Overall, our study exposes two critical gaps: the lack of semantics-preserving bytecode obfuscation tools, and the vulnerability of many detection approaches to adversarial control flow rewriting. Representing bytecode as images, *seeing through obfuscation*, emerges as a promising and structurally resilient direction for detecting evasive SC malware. By providing the first systematic benchmark for evaluating both obfuscation robustness and detection resilience, this work establishes a foundation for developing the next generation of obfuscation-aware analysis techniques in the Ethereum ecosystem.

Chapter 9.

Conclusion

This chapter concludes the dissertation by summarizing its main contributions, outlining future research perspectives, and reflecting on the broader lessons emerging from the work. It highlights how reproducible, data-driven methodologies can bridge the gap between theoretical research and practical deployment across two domains: Decentralized Finance and Smart Contract security.

Chapter Outline

9.1. Thesis Summary

This dissertation explored the design, implementation, and evaluation of Machine Learning (ML) systems across two distinct yet conceptually connected domains: Decentralized Finance (DeFi) and Smart Contract (SC) security. A unifying methodological thread runs through both: the pursuit of reproducible, data-driven frameworks that transform theoretical insights into practical, operational tools.

The first part examined the dynamics of cryptocoin markets and the use of correlation-driven models for price prediction. Building on empirical evidence that digital assets exhibit persistent co-movements, we proposed and validated forecasting techniques that leverage correlated coins as predictive variables. These methods were consolidated into CryptoAnalytics, a fully operational toolkit that automates the forecasting pipeline, from data acquisition and preprocessing to model training, evaluation, and deployment. By integrating Gradient-Boosting Machines (GBMs) and Recurrent Neural Networks (RNNs) within a modular and extensible architecture, CryptoAnalytics bridges the gap between research reproducibility and real-world applicability, supporting both academic experimentation and production-grade forecasting services.

The second part of the dissertation shifted focus to the security of decentralized ecosystems. Here, the attention moved from financial time series to the bytecode of SCs, which underpin Decentralized Applications (dApps) and digital asset management. We introduced PhishingHook, the first reproducible framework for detecting phishing contracts on the Ethereum blockchain through opcode-level analysis. PhishingHook unified dataset construction, bytecode disassembly, model benchmarking, and statistical post hoc validation, demonstrating that static opcode representations can achieve high detection accuracy without relying on dynamic traces or user data. The study further analyzed the trade-offs among model complexity, scalability, and interpretability, providing a solid empirical foundation for proactive SC malware detection.

Building on these foundations, Chapter 8 presented the first systematic evaluation of Ethereum Virtual Machine (EVM) bytecode obfuscation and its impact on SC malware detection. The study revealed that existing obfuscators vary widely in their ability to preserve syntactic validity and functional behavior, and that many detection approaches degrade sharply when control flow is heavily rewritten. At the same time, the results highlighted the resilience of vision-based models, which remain effective even under aggressive structural distortion. This chapter provides a rigorous empirical basis for understanding adversarial bytecode transformations and for guiding the design of more robust, obfuscation-aware detection techniques.

Taken together, these contributions trace a coherent progression: from predictive modeling of financial systems, through the detection of malicious contract behaviors, to a deeper understanding of adversarial transformations in decentralized execution environments. Each part of the dissertation demonstrates how ML-driven methodologies, grounded in transparency and reproducibility, can address complex, high-stakes challenges in blockchain-based systems.

9.2. Research Perspectives and Future Directions

Each chapter of this dissertation concluded with specific directions for continuation. Taken together, these perspectives outline a coherent research trajectory that extends the methodological, experimental, and system-level contributions developed throughout the work.

In the financial forecasting domain, the next steps identified for CryptoAnalytics focus on expanding data coverage and improving operational integration. Future developments include incorporating additional market data sources, exploring learning algorithms beyond GBMs and RNNs, and extending the forecasting pipeline toward real-time and streaming architectures. Such enhancements would broaden the applicability of the toolkit to use cases such as high-frequency trading, market surveillance, and risk monitoring. More generally, the workflow of CryptoAnalytics could be adapted to other time-series domains where co-movement patterns play a central role, including healthcare, energy systems, and environmental monitoring.

In the SC security domain, PhishingHook established the first reproducible framework for opcode-level phishing detection. Future directions emerging from this work include transitioning from offline analysis to real-time detection, enabling malicious contracts to be flagged prior to deployment on-chain. Another line of work involves integrating scalable inference pipelines suitable for production-grade blockchain monitoring platforms, potentially in collaboration with security firms and explorers such as Etherscan. Finally, the methodology underlying PhishingHook can be naturally extended beyond phishing to other classes of malicious contracts, including Ponzi schemes and honeypots.

Beyond malware detection, the findings of Chapter 8 highlight several immediate research directions related to adversarial robustness. The systematic evaluation of BOSC and EVeilM showed that current obfuscation techniques rarely preserve both syntactic validity and functional behavior, and that many existing detectors fail once control flow is heavily rewritten. Future work should therefore focus on designing semantics-preserving obfuscators, developing feature representations that remain stable under structural distortion, and expanding robustness benchmarks to better capture realistic adversarial transformations. The strong resilience of vision-based encodings observed in this dissertation further suggests a promising direction for building more adaptive and obfuscation-tolerant malware detectors.

Building on these results, a broader research perspective naturally emerges: the need for unified threat detection methodologies that generalize beyond a single execution environment. This motivation underlies ScamDetect [290], a vision for cross-runtime and semantically robust SC malware analysis. Rather than defining a concrete system, ScamDetect outlines a forward-looking roadmap that consolidates insights from PhishingHook and the obfuscation study, emphasizing two main objectives: *(i)* integrating complementary bytecode representations (statistical, structural, and semantic), and *(ii)* developing learning models capable of generalizing across heterogeneous execution environments such as the EVM and WebAssembly (WASM). This perspective identifies a long-term research direction toward more adaptable, explainable, and platform-agnostic SC security analysis.

Finally, from a system and deployment perspective, future work includes extending the evaluation of model serving infrastructures used throughout this dissertation. Beyond latency, ad-

ditional metrics such as throughput, resource utilization, and security guarantees should be considered. Confidential ML scenarios are of particular interest, especially in sensitive domains such as finance, healthcare, and defense, where privacy-preserving inference is essential. Moreover, broadening the evaluation to include models trained and deployed in alternative ecosystems, such as PyTorch, JAX [291], and Caffe [292], would enable a more comprehensive understanding of trade-offs across serving frameworks.

Overall, the future directions outlined in this dissertation follow a consistent trajectory: from operationalizing correlation-driven financial forecasting, to automating and scaling SC malware detection, and toward resilient, explainable, and cross-platform analysis. Each step reinforces the central philosophy of this work: combining reproducibility, transparency, and adaptability to translate data-driven research into deployable, high-impact systems.

9.3. Takeaways and Concluding Remarks

The works presented in this dissertation share a common methodological foundation: the systematic design of reproducible frameworks that transform empirical insights into practical tools. Across domains as diverse as financial forecasting and SC security, several overarching lessons emerged.

From the development of CryptoAnalytics, three main insights were gained. First, although cryptocoin markets are highly volatile, their dynamics are not entirely random. Persistent comovement and correlation patterns can be identified and systematically leveraged to produce reliable forecasts. Second, while advanced models such as GBMs and RNNs deliver strong predictive performance, their complexity often limits accessibility and reproducibility. By encapsulating these workflows into an integrated and configurable toolkit, CryptoAnalytics showed that robust forecasting can remain both transparent and practical. Third, deployment experiments confirmed that such models can be efficiently served through modern inference frameworks, demonstrating that research-grade tools can evolve into production-ready forecasting systems.

From the study of phishing detection in Ethereum, embodied by PhishingHook, further lessons were drawn. Static opcode-level representations proved sufficient to identify malicious contracts with high accuracy, without relying on dynamic transaction data. The framework's modular design enabled extensive benchmarking and statistical validation, revealing that simple histogram-based classifiers remain competitive with more complex architectures. Additional experiments showed that although large models such as Transformers and vision-based encoders do not yet surpass simpler classifiers in accuracy, they scale more effectively as datasets expand. Time-resistance analyses further indicated that static models maintain stable detection performance over time, underscoring their practical relevance for real-world monitoring systems.

Chapter 8 extended these insights by providing the first systematic evaluation of EVM bytecode obfuscation and its implications for malware detection. The study demonstrated that existing obfuscators differ sharply in their ability to preserve structural validity and contract semantics,

and that many ML-based detectors, particularly sequence and graph-driven models, are highly sensitive to adversarial control-flow rewriting. At the same time, the results highlighted the resilience of vision-based approaches, which maintain strong performance even under aggressive structural distortion. Together, these findings clarify the trade-offs between obfuscation strength, semantic preservation, and detection robustness in adversarial blockchain environments.

Overall, this dissertation demonstrates that reproducible ML frameworks can effectively bridge theoretical research and deployable applications. In DeFi, this approach enables transparent, data-driven forecasting grounded in empirical market structure. In blockchain security, it enables early, reliable, and privacy-preserving detection of malicious SCs through static analysis alone.

Both domains converge on a common vision: the creation of open, interpretable, and adaptive intelligent systems capable of operating within complex, high-stakes environments. The contributions presented here thus mark not an endpoint but a foundation, a step toward continuously evolving, trustworthy analytics for decentralized and data-intensive ecosystems.

References

[1] Christopher M. Bishop. 2007. *Pattern recognition and machine learning, 5th Edition*. *Information science and statistics*. Springer. ISBN: 9780387310732. https://www.worldcat.org/oclc/71008143.

[2] Johannes Kepler. 1609. *Astronomia nova seu physica coelestis, tradita commentariis de motibus stellae Martis ex observationibus G. V. Tychonis Brahe*. Heidelberg, Voegelin. https://doi.org/10.3931/e-rara-558.

[3] Johann Balmer. 1885. Notiz über die spektrallinien des wasserstoffs. *Annalen der Physik und Chemie*, 261, 5, 80–87. https://doi.org/10.1002/andp.18852610506.

[4] Johannes Rydberg. 1890. Recherches sur la constitution des spectres d'émission des éléments chimiques. 23, 1–92. https://tinyurl.com/rydberg-1890.

[5] Niels Bohr. 1913. On the constitution of atoms and molecules. *Philosophical Magazine*, 26, 1–25. Series 6. https://doi.org/10.1080/14786441308634955.

[6] Yehuda Koren, Robert M. Bell and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer*, 42, 8, 30–37. DOI: 10.1109/MC.2009.263.

[7] Yongmao Yang, Kampol Woradit and Kenneth Cosh. 2025. Hybrid movie recommendation system with user partitioning and log likelihood content comparison. *IEEE Access*, 13, 11609–11622. DOI: 10.1109/ACCESS.2025.3529515.

[8] Xueyan Yin, Genze Wu, Jinze Wei, Yanming Shen, Heng Qi and Baocai Yin. 2022. Deep learning on traffic prediction: methods, analysis, and future directions. *IEEE Trans. Intell. Transp. Syst.*, 23, 6, 4927–4943. DOI: 10.1109/TITS.2021.3054840.

[9] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang and Jianzhong Qi. 2020. GMAN: A graph multi-attention network for traffic prediction. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 1234–1241. DOI: 10.1609/AAAI.V34I01.5477.

[10] Pavani Chitrapu, Mahesh Kumar Morampudi and Hemantha Kumar Kalluri. 2025. Robust face recognition using deep learning and ensemble classification. *IEEE Access*, 13, 99957–99969. DOI: 10.1109/ACCESS.2025.3575192.

[11] Chi Nhan Duong, Kha Gia Quach, Ibsa Jalata, Ngan Le and Khoa Luu. 2019. Mobiface: A lightweight deep learning face recognition on mobile devices. In *10th IEEE International Conference on Biometrics Theory, Applications and Systems, BTAS 2019, Tampa, FL, USA, September 23-26, 2019*. IEEE, 1–6. DOI: 10.1109/BTAS46853.2019.9185981.

[12] Vijay Kumar Vishwakarma and Narayan P. Bhosale. 2025. A survey of recent machine learning techniques for stock prediction methodologies. *Neural Comput. Appl.*, 37, 4, 1951–1972. DOI: 10.1007/S00521-024-10867-Y.

[13] Siddharth Bhatore, Lalit Mohan S. and Y. Raghu Reddy. 2020. Machine learning techniques for credit risk evaluation: a systematic literature review. *J. Bank. Financial Technol.*, 4, 1, 111–138. DOI: 10.1007/S42786-020-00020-3.

[14] Safak Kayikci and Taghi M. Khoshgoftaar. 2024. Blockchain meets machine learning: a survey. *J. Big Data*, 11, 1, 9. DOI: 10.1186/S40537-023-00852-Y.

[15] Bingqiao Luo, Zhen Zhang, Qian Wang, Anli Ke, Shengliang Lu and Bingsheng He. 2025. Ai-powered fraud detection in decentralized finance: A project life cycle perspective. *ACM Comput. Surv.*, 57, 4, 96:1–96:38. DOI: 10.1145/3705296.

[16] Fabian Schär. 2021. Decentralized finance: on blockchain- and smart contract-based financial markets. *Fed. Reserve Bank St. Louis Rev.*, 103, 2, 153–174. DOI: 10.20955/r.103.153-74.

[17] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system. (2008). https://bitcoin.org/bitcoin.pdf.

[18] Gavin Wood. Ethereum: a secure decentralised generalised transaction ledger. (2025). https://ethereum.github.io/yellowpaper/paper.pdf.

[19] Nick Szabo. 1997. Formalizing and securing relationships on public networks. *First Monday*, 2, 9. DOI: 10.5210/FM.V2I9.548.

[20] Tascha Che. 2022. Public blockchains are the new national economies of the metaverse. Accessed on 2025-09-01. https://www.wired.com/story/blockchain-cryptocurrency-economics/.

[21] Jiaqi Liang, Linjing Li, Weiyun Chen and Daniel Zeng. 2019. Towards an understanding of cryptocurrency: A comparative analysis of cryptocurrency, foreign exchange, and stock. In *2019 IEEE International Conference on Intelligence and Security Informatics, ISI 2019, Shenzhen, China, July 1-3, 2019*. IEEE, 137–139. DOI: 10.1109/ISI.2019.8823373.

[22] Srinivasan Palamalai, K. Krishna Kumar and Bipasha Maity. 2021. Testing the random walk hypothesis for leading cryptocurrencies. *Borsa Istanbul Review*, 21, 3, 256–268. DOI: 10.1016/j.bir.2020.10.006.

[23] Paraskevi Katsiampa. 2019. Volatility co-movement between bitcoin and ether. *Finance Research Letters*, 30, 221–227. DOI: 10.1016/j.frl.2018.10.005.

[24] Nektarios Aslanidis, Aurelio F. Bariviera and Oscar Martínez-Ibañez. 2019. An analysis of cryptocurrencies conditional cross correlations. *Finance Research Letters*, 31, 130–137. DOI: 10.1016/j.frl.2019.04.019.

[25] Nicola Atzei, Massimo Bartoletti and Tiziana Cimoli. 2017. A survey of attacks on ethereum smart contracts (sok). In *Principles of Security and Trust - 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings* (Lecture Notes in Computer Science). Matteo Maffei and Mark Ryan, editors. Vol. 10204. Springer, 164–186. DOI: 10.1007/978-3-662-54455-6_8.

[26] Huashan Chen, Marcus Pendleton, Laurent Njilla and Shouhuai Xu. 2021. A survey on ethereum systems security: vulnerabilities, attacks, and defenses. *ACM Comput. Surv.*, 53, 3, 67:1–67:43. DOI: 10.1145/3391195.

[27] Daniel Perez and Benjamin Livshits. 2021. Smart contract vulnerabilities: vulnerable does not imply exploited. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. Michael D. Bailey and Rachel Greenstadt, editors. USENIX Association, 1325–1341. https://www.usenix.org/conference/usenixsecurity21/presentation/perez.

[28] Tengyun Jiao, Zhiyu Xu, Minfeng Qi, Sheng Wen, Yang Xiang and Gary Nan. 2024. A survey of ethereum smart contract security: attacks and detection. *Distributed Ledger Technol. Res. Pract.*, 3, 3, 23:1–23:28. DOI: 10.1145/3643895.

[29] David Siegel. 2016. Understanding the dao attack. Accessed on 2025-09-01. https://www.coindesk.com/learn/understanding-the-dao-attack/.

[30] TRM Labs. 2021. Grim finance hacked: 600 million in crypto stolen in december. Accessed on 2025-09-01. https://www.trmlabs.com/post/grim-finance-hacked-600-million-in-crypto-stolen-in-december.

[31] Pengcheng Zhang, Qifan Yu, Yan Xiao, Hai Dong, Xiapu Luo, Xiao Wang and Meng Zhang. 2023. Bian: smart contract source code obfuscation. *IEEE Trans. Software Eng.*, 49, 9, 4456–4476. DOI: 10.1109/TSE.2023.3298609.

[32] Qifan Yu, Pengcheng Zhang, Hai Dong, Yan Xiao and Shunhui Ji. 2022. Bytecode obfuscation for smart contracts. In *29th Asia-Pacific Software Engineering Conference, APSEC 2022, Virtual Event, Japan, December 6-9, 2022*. IEEE, 566–567. DOI: 10.1109/APSEC57359.2022.00083.

[33] Titouan Forissier. 2024. Eveilm: evm bytecode obfuscation (dissertation). Retrieved from: https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-359674. Dissertation. (2024).

[34] Zekun Hou, Changtong Dong and Ying Shang. 2023. Hermhd: enhancing smart contract security based on code obfuscation. In *Proceedings of the 2023 11th International Conference on Information Technology: IoT and Smart City, ICIT 2023, Kyoto, Japan, December 14-17, 2023*. ACM, 96–101. DOI: 10.1145/3638985.3639001.

[35] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou and Kilian Q. Weinberger, editors, 1106–1114. https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

[36] Volodymyr Mnih, Nicolas Heess, Alex Graves and Koray Kavukcuoglu. 2014. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence and Kilian Q. Weinberger, editors, 2204–2212. https://proceedings.neurips.cc/paper/2014/hash/09c6c3783b4a70054da74f2538ed47c6-Abstract.html.

[37] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J. Franklin, Joseph E. Gonzalez and Ion Stoica. 2017. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*. Aditya Akella and Jon Howell, editors. USENIX Association, 613–627. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw.

[38] Mauro Ribeiro, Katarina Grolinger and Miriam A. M. Capretz. 2015. Mlaas: machine learning as a service. In *14th IEEE International Conference on Machine Learning and Applications, ICMLA 2015, Miami, FL, USA, December 9-11, 2015*. Tao Li, Lukasz A. Kurgan, Vasile Palade, Randy Goebel, Andreas Holzinger, Karin Verspoor and M. Arif Wani, editors. IEEE, 896–902. DOI: 10.1109/ICMLA.2015.152.

[39] Robert Philipp, Andreas Mladenow, Christine Strauss and Alexander Völz. 2020. Machine learning as a service: challenges in research and applications. In *iiWAS '20: The 22nd International Conference on Information Integration and Web-based Applications & Services, Virtual Event / Chiang Mai, Thailand, November 30 - December 2, 2020*. Maria Indrawan-Santiago, Eric Pardede, Ivan Luiz Salvadori, Matthias Steinbauer, Ismail Khalil and Gabriele Kotsis, editors. ACM, 396–406. DOI: 10.1145/3428757.3429152.

[40] Mazin Yousif. 2017. Intelligence in the cloud - we need a lot of it. *IEEE Cloud Comput.*, 4, 6, 4–6. DOI: 10.1109/MCC.2018.1081057.

[41] 2025. Amazon sagemaker pricing. Accessed on 2025-09-01. https://aws.amazon.com/sagemaker/pricing/.

[42] 2025. Azure machine learning pricing. Accessed on 2025-09-01. https://azure.microsoft.com/en-us/pricing/details/machine-learning/.

[43] 2025. Google cloud ai pricing. Accessed on 2025-09-01. https://cloud.google.com/ai-platform/pricing.

[44] Nicolas Papernot, Patrick D. McDaniel, Arunesh Sinha and Michael P. Wellman. 2018. Sok: security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*. IEEE, 399–414. DOI: 10.1109/EUROSP.2018.00035.

[45] Sagar Sharma and Keke Chen. 2021. Confidential machine learning on untrusted platforms: a survey. *Cybersecur.*, 4, 1, 30. DOI: 10.1186/S42400-021-00092-8.

[46] Google Developers. 2025. Tensorflow serving documentation. Accessed on 2025-09-01. https://www.tensorflow.org/tfx/guide/serving.

[47] PyTorch Developers. 2025. Torchserve documentation. Accessed on 2025-09-01. https://pytorch.org/serve/.

[48] BentoML Developers. 2025. Bentoml documentation. Accessed on 2025-09-01. https://docs.bentoml.com/en/latest/.

[49] MLflow Developers. 2025. Mlflow documentation. Accessed on 2025-09-01. https://mlflow.org/docs/latest/index.html.

[50] Seldon Technologies Developers. 2025. Mlserver documentation. Accessed on 2025-09-01. https://mlserver.readthedocs.io/en/stable/.

[51] Pasquale De Rosa, Yérom-David Bromberg, Pascal Felber, Djob Mvondo and Valerio Schiavoni. 2024. On the cost of model-serving frameworks: an experimental evaluation. In *IEEE International Conference on Cloud Engineering, IC2E 2024, Paphos, Cyprus, September 24-27, 2024*. IEEE, 221–232. DOI: 10.1109/IC2E61754.2024.00032.

[52] Pasquale De Rosa and Valerio Schiavoni. 2022. Understanding cryptocoins trends correlations. In *Distributed Applications and Interoperable Systems: 22nd IFIP WG 6.1 International Conference, DAIS 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings* (Lecture Notes in Computer Science). David M. Eyers and Spyros Voulgaris, editors. Vol. 13272. Springer, 29–36. DOI: 10.1007/978-3-031-16092-9_3.

[53] Pasquale De Rosa, Pascal Felber and Valerio Schiavoni. 2023. Practical forecasting of cryptocoins timeseries using correlation patterns. In *Proceedings of the 17th ACM International Conference on Distributed and Event-based Systems, DEBS 2023, Neuchâtel, Switzerland, June 27-30, 2023*. Valerio Schiavoni, Marcelo Pasin, Bettina Kemme and Etienne Rivière, editors. ACM, 80–90. DOI: 10.1145/3583678.3596888.

[54] Pasquale De Rosa, Pascal Felber and Valerio Schiavoni. 2024. Cryptoanalytics: cryptocoins price forecasting with machine learning techniques. *SoftwareX*, 26, 101663. DOI: 10.1016/J.SOFTX.2024.101663.

[55] Pasquale De Rosa, Simon Queyrut, Yérom-David Bromberg, Pascal Felber and Valerio Schiavoni. 2025. Phishinghook: catching phishing ethereum smart contracts leveraging EVM opcodes. In *55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2025, Naples, Italy, June 23-26, 2025*. IEEE, 222–232. DOI: 10.1109/DSN64029.2025.00033.

[56] Pasquale De Rosa, Simon Queyrut, Yérom-David Bromberg, Pascal Felber and Valerio Schiavoni. 2025. Phishinghook: catching phishing ethereum smart contracts leveraging EVM opcodes. In *2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2025 - Supplemental Volume, Naples, Italy, June 23-26, 2025*. IEEE, 265–266. DOI: 10.1109/DSN-S65789.2025.00075.

[57] Pasquale De Rosa, Pascal Felber and Valerio Schiavoni. 2026. Seeing through EVM bytecode obfuscation. *IEEE Access*. DOI: 10.1109/ACCESS.2026.3662238.

[58] Pasquale De Rosa, Pascal Felber and Valerio Schiavoni. 2023. Cryptocoins analytics: analysis and forecasting of cryptocurrency price trends. Accessed on 2025-09-01. https://github.com/quapsale/cryptoanalytics.

[59] Pasquale De Rosa, Pascal Felber and Valerio Schiavoni. 2024. Cryptoanalytics: cryptocoins price forecasting with machine learning techniques. Accessed on 2025-09-01. https://github.com/quapsale/cryptoanalytics-software.

[60] Pasquale De Rosa, Yérom-David Bromberg, Pascal Felber, Djob Mvondo and Valerio Schiavoni. 2024. On the cost of model-serving frameworks: an experimental evaluation. Accessed on 2025-09-01. https://doi.org/10.5281/zenodo.11200983.

[61] Pasquale De Rosa, Simon Queyrut, Yérom-David Bromberg, Pascal Felber and Valerio Schiavoni. 2025. Phishinghook: catching phishing ethereum smart contracts leveraging EVM opcodes. Accessed on 2025-09-01. https://doi.org/10.5281/zenodo.14933708.

[62] Pasquale De Rosa, Pascal Felber and Valerio Schiavoni. 2026. Seeing through EVM bytecode obfuscation. Accessed on 2026-01-01. https://doi.org/10.5281/zenodo.15660760.

[63] Ian J. Goodfellow, Yoshua Bengio and Aaron C. Courville. 2016. *Deep Learning*. *Adaptive computation and machine learning*. MIT Press. ISBN: 978-0-262-03561-3. http://www.deeplearningbook.org/.

[64] Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani. 2013. *An Introduction to Statistical Learning: with Applications in R*. *Springer Texts in Statistics*. Springer. ISBN: 978-1-4614-7137-0. DOI: 10.1007/978-1-4614-7138-7.

[65] Tom M. Mitchell. 1997. *Machine learning, International Edition*. *McGraw-Hill Series in Computer Science*. McGraw-Hill. ISBN: 978-0-07-042807-2. https://www.worldcat.org/oclc/61321007.

[66] Francis Galton. 1886. Regression towards mediocrity in hereditary stature. *The Journal of the Anthropological Institute of Great Britain and Ireland*, 15, 246–263. http://www.jstor.org/stable/2841583.

[67] Carl Friedrich Gauss. 1809. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*. Hamburgi: sumtibus Frid. Perthes et I. H. Besser. https://doi.org/10.3931/e-rara-522.

[68] David A. Belsley, Edwin Kuh and Roy E. Welsch. 1980. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*. *Wiley Series in Probability and Statistics*. John Wiley & Sons. ISBN: 978-0-471-05856-4. DOI: 10.1002/0471725153.

[69] Peter McCullagh and John A. Nelder. 1989. *Generalized Linear Models*. (2nd ed.). Routledge. ISBN: 978-0-412-31760-6. DOI: 10.1201/9780203753736.

[70] David R. Cox. 1958. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20, 2, 215–232. DOI: 10.1111/j.2517-6161.1958.tb00292.x.

[71] Pierre-François Verhulst. 1838. Notice sur la loi que la population poursuit dans son accroissement. *Correspondance Mathématique et Physique*, 10, 113–121.

[72] Ronald A. Fisher. 1922. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 222, 309–368. DOI: 10.1098/rsta.1922.0009.

[73] Peter J. Green. 1984. Iteratively reweighted least squares for maximum likelihood estimation, and some robust and resistant alternatives. *Journal of the Royal Statistical Society: Series B (Methodological)*, 46, 2, 149–170. DOI: 10.1111/j.2517-6161.1984.tb01288.x.

[74] J. C. Stoltzfus. 2011. Logistic regression: a brief primer. *Academic Emergency Medicine*, 18, 10, 1099–1104. DOI: 10.1111/j.1553-2712.2011.01185.x.

[75] Thomas M. Cover and Peter E. Hart. 1967. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory*, 13, 1, 21–27. DOI: 10.1109/TIT.1967.1053964.

[76] Stuart Geman, Elie Bienenstock and René Doursat. 1992. Neural networks and the bias/variance dilemma. *Neural Comput.*, 4, 1, 1–58. DOI: 10.1162/NECO.1992.4.1.1.

[77] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Mach. Learn.*, 20, 3, 273–297. DOI: 10.1007/BF00994018.

[78] David S. Broomhead and David Lowe. 1988. Multivariable functional interpolation and adaptive networks. *Complex Syst.*, 2, 3. http://www.complex-systems.com/abstracts/v02_i03_a05.html.

[79] James N. Morgan and John A. Sonquist. 1963. Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, 58, 302, 415–434. DOI: 10.1080/01621459.1963.10500855.

[80] Leo Breiman. 2001. Random forests. *Mach. Learn.*, 45, 1, 5–32. DOI: 10.1023/A:1010933404324.

[81] Leo Breiman, J. H. Friedman, Richard A. Olshen and C. J. Stone. 1984. *Classification and Regression Trees*. Wadsworth. ISBN: 0-534-98053-8.

[82] Leo Breiman. 1996. Bagging predictors. *Mach. Learn.*, 24, 2, 123–140. DOI: 10.1007/BF00058655.

[83] Yoav Freund and Robert E. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55, 1, 119–139. DOI: 10.1006/JCSS.1997.1504.

[84] Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani. 2004. Least angle regression. *Annals of Statistics*, 32, 2, (Apr. 2004), 407–499. DOI: 10.1214/009053604000000067.

[85] Corrado Gini. 1912. *Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche*. Tipografia di Paolo Cuppini, Bologna, Italy. https://books.google.ch/books?id=fqjaBPMxB9kC.

[86] Claude E. Shannon. 1948. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27, 4, 623–656. DOI: 10.1002/J.1538-7305.1948.TB00917.X.

[87] Ronald A. Fisher. 1936. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7, 2, 179–188. DOI: 10.1111/j.1469-1809.1936.tb02137.x.

[88] Jerome H. Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 29, 5, 1189–1232. DOI: 10.1214/aos/1013203451.

[89] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen and Rajeev Rastogi, editors. ACM, 785–794. DOI: 10.1145/2939672.2939785.

[90] Kaggle. 2014. Higgs boson machine learning challenge. Accessed on 2025-09-12. https://www.kaggle.com/c/higgs-boson/discussion/10998.

[91] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan and Roman Garnett, editors, 3146–3154. https://proceedings.neurips.cc/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html.

[92] Liudmila Ostroumova Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush and Andrey Gulin. 2018. Catboost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi and Roman Garnett, editors, 6639–6649. https://proceedings.neurips.cc/paper/2018/hash/14491b756b3a51daac41c24863285549-Abstract.html.

[93] Frank Rosenblatt. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65, 6, 386–408. DOI: 10.1037/h0042519.

[94] Kurt Hornik, Maxwell Stinchcombe and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2, 5, 359–366. DOI: 10.1016/0893-6080(89)90020-8.

[95] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*. Johannes Fürnkranz and Thorsten Joachims, editors. Omnipress, 807–814. https://icml.cc/Conferences/2010/papers/432.pdf.

[96] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard and Lawrence D. Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1, 4, 541–551. DOI: 10.1162/NECO.1989.1.4.541.

[97] Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *Annals of Mathematical Statistics*, 22, 3, (Sept. 1951), 400–407. DOI: 10.1214/aoms/1177729586.

[98] David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature*, 323, 6088, (Oct. 1986), 533–536. DOI: 10.1038/323533a0.

[99] Kunihiko Fukushima. 1980. Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36, 193–202. DOI: 10.1007/BF00344251.

[100] Mingxing Tan and Quoc V. Le. 2019. Efficientnet: rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA* (Proceedings of Machine Learning Research). Kamalika Chaudhuri and Ruslan Salakhutdinov, editors. Vol. 97. PMLR, 6105–6114. http://proceedings.mlr.press/v97/tan19a.html.

[101] Qilong Wang, Banggu Wu, Pengfei Zhu, Peihua Li, Wangmeng Zuo and Qinghua Hu. 2020. Eca-net: efficient channel attention for deep convolutional neural networks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 11531–11539. DOI: 10.1109/CVPR42600.2020.01155.

[102] Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science*, 14, 2, 179–211. DOI: 10.1207/s15516709cog1402_1.

[103] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9, 8, 1735–1780. DOI: 10.1162/NECO.1997.9.8.1735.

[104] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. Alessandro Moschitti, Bo Pang and Walter Daelemans, editors. ACL, 1724–1734. DOI: 10.3115/V1/D14-1179.

[105] Yoshua Bengio, Patrice Y. Simard and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, 5, 2, 157–166. DOI: 10.1109/72.279181.

[106] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan and Roman Garnett, editors, 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[107] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit and Neil Houlsby. 2021. An image is worth 16x16 words: transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=YicbFdNTTy.

[108] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1, 8, 9.

[109] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21, 140:1–140:67. https://jmlr.org/papers/v21/20-074.html.

[110] Andreas M. Antonopoulos. 2014. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. O'Reilly Media. ISBN: 978-1-4493-7402-5.

[111] Andreas M. Antonopoulos and Gavin Wood. 2018. *Mastering Ethereum: Building Smart Contracts and DApps*. O'Reilly Media. ISBN: 978-1-4919-7212-3.

[112] Juan A. Garay, Aggelos Kiayias and Nikos Leonardos. 2015. The bitcoin backbone protocol: analysis and applications. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II* (Lecture Notes in Computer Science). Elisabeth Oswald and Marc Fischlin, editors. Vol. 9057. Springer, 281–310. DOI: 10.1007/978-3-662-46803-6_10.

[113] Ralph C. Merkle. 1987. A digital signature based on a conventional encryption function. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings* (Lecture Notes in Computer Science). Carl Pomerance, editor. Vol. 293. Springer, 369–378. DOI: 10.1007/3-540-48184-2_32.

[114] Donald R. Morrison. 1968. PATRICIA - practical algorithm to retrieve information coded in alphanumeric. *J. ACM*, 15, 4, 514–534. DOI: 10.1145/321479.321481.

[115] Leslie Lamport, Robert E. Shostak and Marshall C. Pease. 1982. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4, 3, 382–401. DOI: 10.1145/357172.357176.

[116] Sunny King and Scott Nadal. Ppcoin: peer-to-peer crypto-currency with proof-of-stake. (2012). https://peercoin.net/assets/paper/peercoin-paper.pdf.

[117] Vitalik Buterin and Virgil Griffith. 2017. Casper the friendly finality gadget. *CoRR*. http://arxiv.org/abs/1710.09437 arXiv: 1710.09437.

[118] Guinness World Records. 2009. First bitcoin transaction. Accessed on 2025-09-18. https://www.guinnessworldrecords.com/world-records/696243-first-bitcoin-transaction.

[119] Ethereum Foundation. 2025. Erc-20 token standard. Accessed on 2025-09-18. https://ethereum.org/developers/docs/standards/tokens/erc-20/.

[120] Charlie Lee. Litecoin: a peer-to-peer internet currency. (2011). https://litecoin.org.

[121] Aggelos Kiayias, Alexander Russell, Bernardo David and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I* (Lecture Notes in Computer Science). Jonathan Katz and Hovav Shacham, editors. Vol. 10401. Springer, 357–388. DOI: 10.1007/978-3-319-63688-7_12.

[122] David Schwartz, Noah Youngs and Arthur Britto. The ripple protocol consensus algorithm. (2014). https://ripple.com/files/ripple_consensus_whitepaper.pdf.

[123] Binance Team. Binance coin (bnb) whitepaper. (2017). https://www.exodus.com/assets/docs/binance-coin-whitepaper.pdf.

[124] Tether Team. Tether: fiat currencies on the bitcoin blockchain. (2014). https://tinyurl.com/4pjds8vs.

[125] USDC Team. Usd coin (usdc) whitepaper. (2018). https://cryptoactu.com/wp-content/uploads/2018/12/WhitePaper.pdf.

[126] International Monetary Fund. 2023. World economic outlook. Accessed on 2025-09-18. https://www.imf.org/en/Publications/WEO.

[127] Coinbase. 2025. Coinbase exchange. Accessed on 2025-09-18. https://www.coinbase.com.

[128] Kraken. 2025. Kraken exchange. Accessed on 2025-09-18. https://www.kraken.com.

[129] Binance. 2025. Binance exchange. Accessed on 2025-09-18. https://www.binance.com.

[130] Uniswap. 2025. Uniswap exchange. Accessed on 2025-09-18. https://www.uniswap.org.

[131] Eugene F. Fama. 1965. Random walks in stock market prices. *Financial Analysts Journal*, 21, 5, 55–59. Retrieved 8 May 2022 from http://www.jstor.org/stable/4469865.

[132] Coinbase. 2025. Price page information. Accessed on 2025-09-18. https://help.coinbase.com/en/coinbase/getting-started/crypto-education/price-page-information-.

[133] Mehdi Salehi, Jeremy Clark and Mohammad Mannan. 2022. Not so immutable: upgradeability of smart contracts on ethereum. In *Financial Cryptography and Data Security. FC 2022 International Workshops - CoDecFin, DeFi, Voting, WTSC, Grenada, May 6, 2022, Revised Selected Papers* (Lecture Notes in Computer Science). Shin'ichiro Matsuo, Lewis Gudgeon, Ariah Klages-Mundt, Daniel Perez Hernandez, Sam Werner, Thomas Haines, Aleksander Essex, Andrea Bracciali and Massimiliano Sala, editors. Vol. 13412. Springer, 539–554. DOI: 10.1007/978-3-031-32415-4_33.

[134] Tahrim Hossain, Sakib Hassan, Faisal Haque Bappy, Muhammad Nur Yanhaona, Tarannum Shaila Zaman and Tariqul Islam. 2025. Bridging immutability with flexibility: A scheme for secure and efficient smart contract upgrades. *CoRR*, abs/2504.09652. arXiv: 2504.09652. DOI: 10.48550/ARXIV.2504.09652.

[135] Solidity Team. 2025. Solidity language documentation. Accessed on 2025-09-18. https://soliditylang.org/.

[136] Vyper Project. 2025. Vyper language documentation. Accessed on 2025-09-18. https://docs.vyperlang.org/en/stable/.

[137] EVM.codes. 2025. Evm opcodes (shanghai update). Accessed on 2025-09-22. https://www.evm.codes/?fork=shanghai.

[138] Christof Ferreira Torres, Mathis Steichen and Radu State. 2019. The art of the scam: demystifying honeypots in ethereum smart contracts. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*. USENIX Association, 1591–1607. https://www.usenix.org/conference/usenixsecurity19/presentation/ferreira.

[139] Shuhui Fan, Shaojing Fu, Haoran Xu and Xiaochun Cheng. 2021. Al-SPSD: anti-leakage smart ponzi schemes detection in blockchain. *Inf. Process. Manag.*, 58, 4, 102587. DOI: 10.1016/j.ipm.2021.102587.

[140] Arkan Hammoodi Hasan Kabla, Mohammed Anbar, Selvakumar Manickam and Shankar Karupayah. 2022. Eth-PSD: a machine learning-based phishing scam detection approach in ethereum. *IEEE Access*, 10, 118043–118057. DOI: 10.1109/ACCESS.2022.3220780.

[141] Bowen He, Yuan Chen, Zhuo Chen, Xiaohui Hu, Yufeng Hu, Lei Wu, Rui Chang, Haoyu Wang and Yajin Zhou. 2023. Txphishscope: towards detecting and understanding transaction-based phishing on ethereum. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), Copenhagen, Denmark, November 26–30, 2023*. ACM, 120–134. DOI: 10.1145/3576915.3623210.

[142] Christoph Sendner, Huili Chen, Hossein Fereidooni, Lukas Petzi, Jan König, Jasper Stang, Alexandra Dmitrienko, Ahmad-Reza Sadeghi and Farinaz Koushanfar. 2023. Smarter contracts: detecting vulnerabilities in smart contracts with deep transfer learning. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. Internet Society. DOI: 10.14722/ndss.2023.23263.

[143] Seon-Jin Hwang, Seok-Hwan Choi, Jinmyeong Shin and Yoon-Ho Choi. 2022. Codenet: code-targeted convolutional neural network architecture for smart contract vulnerability detection. *IEEE Access*, 10, 32595–32607. DOI: 10.1109/ACCESS.2022.3162065.

[144] Samuel Banning Osei, Zhongchen Ma and Rubing Huang. 2024. Smart contract vulnerability detection using wide and deep neural network. *Sci. Comput. Program.*, 238, 103172. DOI: 10.1016/j.scico.2024.103172.

[145] Peng Qian, Zhenguang Liu, Qinming He, Roger Zimmermann and Xun Wang. 2020. Towards automated reentrancy detection for smart contracts based on sequential models. *IEEE Access*, 8, 19685–19695. DOI: 10.1109/ACCESS.2020.2969429.

[146] Peiqiang Li, Guojun Wang, Xiaofei Xing, Jinyao Zhu, Wanyi Gu and Guangxin Zhai. 2024. A smart contract vulnerability detection method based on deep learning with opcode sequences. *Peer-to-Peer Netw. Appl.*, 17, 5, (Sept. 2024), 3222–3238. DOI: 10.1007/s12083-024-01750-7.

[147] Huiwen Hu, Qianlan Bai and Yuedong Xu. 2022. Scsguard: deep scam detection for ethereum smart contracts. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops, INFOCOM 2022 - Workshops, New York, NY, USA, May 2-5, 2022*. IEEE, 1–6. DOI: 10.1109/INFOCOMWKSHPS54753.2022.9798296.

[148] Xuanchen Zhou, Wenzhong Yang, Liejun Wang, Fuyuan Wei, KeZiErBieKe HaiLaTi and Yuanyuan Liao. 2023. The detection of fraudulent smart contracts based on eca-efficientnet and data enhancement. *Computers, Materials and Continua*, 77, 3, 4073–4087. DOI: 10.32604/cmc.2023.040253.

[149] Web3 Foundation / Parity Technologies. 2025. Polkadot: secure, interoperable, scalable multi-chain protocol. Accessed on 2025-09-22. https://polkadot.com/.

[150] NEAR Foundation. 2025. Near: a developer-friendly, sharded layer-1 blockchain with chain abstraction. Accessed on 2025-09-22. https://www.near.org/.

[151] DFINITY Foundation / Internet Computer. 2025. Internet computer: the world computer — secure, network-resident code and data. Accessed on 2025-09-22. https://internetcomputer.org/.

[152] EOS Network Foundation. 2025. Eos network: high-performance blockchain for scalable dapps. Accessed on 2025-09-22. https://eosnetwork.com/.

[153] WebAssembly Community Group / W3C. 2025. Webassembly: efficient, portable, and safe execution for web & beyond. Accessed on 2025-09-22. https://webassembly.org/.

[154] Ethereum WebAssembly (ewasm) Team. 2025. Ethereum webassembly (ewasm): a proposed redesign of the ethereum smart contract execution layer using webassembly. Accessed on 2025-09-22. https://ewasm.readthedocs.io/en/mkdocs/.

[155] Ningyu He, Haoyu Wang, Lei Wu, Xiapu Luo, Yao Guo and Xiangqun Chen. 2025. A survey on EOSIO systems security: vulnerability, attack, and mitigation. *Frontiers Comput. Sci.*, 19, 6, 196806. DOI: 10.1007/S11704-024-3278-Y.

[156] CoinDesk. 2016. Understanding the dao attack. Accessed on 2025-09-22. https://www.coindesk.com/learn/understanding-the-dao-attack.

[157] TRM Labs Investigations. 2021. Grim finance hacked: 600 million in crypto stolen in december. Accessed on 2025-09-22. https://www.trmlabs.com/resources/blog/grim-finance-hacked-600-million-in-crypto-stolen-in-december.

[158] De.Fi. 2023. De.fi rekt report: crypto losses reach $1.95b in 2023. Accessed on 2025-09-22. https://de.fi/blog/de-fi-rekt-report-crypto-losses-reach-1-95b-in-2023.

[159] Christof Ferreira Torres, Mathis Steichen and Radu State. 2019. The art of the scam: demystifying honeypots in ethereum smart contracts. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*. Nadia Heninger and Patrick Traynor, editors. USENIX Association, 1591–1607. https://www.usenix.org/conference/usenixsecurity19/presentation/ferreira.

[160] Phylum Research Team. 2024. Typosquat campaign targeting npm developers. Accessed on 2025-09-22. https://blog.phylum.io/typosquat-campaign-targeting-puppeteer-users/.

[161] Ningyu He, Ruiyi Zhang, Haoyu Wang, Lei Wu, Xiapu Luo, Yao Guo, Ting Yu and Xuxian Jiang. 2021. EOSAFE: security analysis of EOSIO smart contracts. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. Michael D. Bailey and Rachel Greenstadt, editors. USENIX Association, 1271–1288. https://www.usenix.org/conference/usenixsecurity21/presentation/he-ningyu.

[162] Jianfei Zhou and Ting Chen. 2023. WASMOD: detecting vulnerabilities in wasm smart contracts. *IET Blockchain*, 3, 4, 172–181. DOI: 10.1049/BLC2.12029.

[163] Weimin Chen, Zihan Sun, Haoyu Wang, Xiapu Luo, Haipeng Cai and Lei Wu. 2022. WASAI: uncovering vulnerabilities in wasm smart contracts. In *ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, July 18 - 22, 2022*. Sukyoung Ryu and Yannis Smaragdakis, editors. ACM, 703–715. DOI: 10.1145/3533767.3534218.

[164] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu and Xiaoqiang Zheng. 2016. Tensorflow: large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467. http://arxiv.org/abs/1603.04467 arXiv: 1603.04467.

[165] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai and Soumith Chintala. 2019. Pytorch: an imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox and Roman Garnett, editors, 8024–8035. https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[166] Dominik Kreuzberger, Niklas Kühl and Sebastian Hirschl. 2023. Machine learning operations (mlops): overview, definition, and architecture. *IEEE Access*, 11, 31866–31879. DOI: 10.1109/ACCESS.2023.3262138.

[167] Sonia Horchidan, Emmanouil Kritharakis, Vasiliki Kalavri and Paris Carbone. 2022. Evaluating model serving strategies over streaming data. In *DEEM '22: Proceedings of the Sixth Workshop on Data Management for End-To-End Machine Learning Philadelphia, PA, USA, 12 June 2022*. Matthias Boehm, Paroma Varma and Doris Xin, editors. ACM, 4:1–4:5. DOI: 10.1145/3533028.3533308.

[168] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar and Zhihao Jia. 2024. Specinfer: accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*. Rajiv Gupta, Nael B. Abu-Ghazaleh, Madan Musuvathi and Dan Tsafrir, editors. ACM, 932–949. DOI: 10.1145/3620666.3651335.

[169] Foteini Strati, Sara McAllister, Amar Phanishayee, Jakub Tarnawski and Ana Klimovic. 2024. Déjàvu: kv-cache streaming for fast, fault-tolerant generative LLM serving. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net. https://openreview.net/forum?id=AbGbGZFYOD.

[170] Yuxin Wang, Yuhan Chen, Zeyu Li, Zhenheng Tang, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou and Xiaowen Chu. 2024. Towards efficient and reliable LLM serving: A real-world workload study. *CoRR*, abs/2401.17644. arXiv: 2401.17644. DOI: 10.48550/ARXIV.2401.17644.

[171] Keras. 2025. Keras documentation. Accessed on 2025-09-23. https://keras.io/.

[172] Hugging Face. 2025. Hugging face documentation. Accessed on 2025-09-23. https://huggingface.co/.

[173] PyTorch. 2025. Mar file specifics. Accessed on 2025-09-23. https://pytorch.org/serve/FAQs.html#what-is-a-mar-file.

[174] PyTorch. 2025. Torchscript documentation. Accessed on 2025-09-23. https://pytorch.org/docs/stable/jit.html.

[175] ONNX. 2025. Onnx documentation. Accessed on 2025-09-23. https://onnx.ai/onnx/index.html.

[176] Scikit-learn Developers. 2025. Scikit-learn documentation. Accessed on 2025-09-23. https://scikit-learn.org/stable/.

[177] Pallets Projects. 2025. Flask documentation. Accessed on 2025-09-23. https://flask.palletsprojects.com/en/stable/.

[178] ONNX Community. 2025. Tf2onnx documentation. Accessed on 2025-09-23. https://github.com/onnx/tensorflow-onnx.

[179] ENOT-AutoDL. 2025. Onnx2torch documentation. Accessed on 2025-09-23. https://github.com/ENOT-AutoDL/onnx2torch.

[180] Omar N. Elayan and Ahmad M. Mustafa. 2021. Android malware detection using deep learning. In *The 12th International Conference on Ambient Systems, Networks and Technologies (ANT 2021) / The 4th International Conference on Emerging Data and Industry 4.0 (EDI40 2021) / Affiliated Workshops, March 23-26, 2021, Warsaw, Poland* (Procedia Computer Science). Elhadi M. Shakshuki and Ansar-Ul-Haque Yasar, editors. Vol. 184. Elsevier, 847–852. DOI: 10.1016/J.PROCS.2021.03.106.

[181] Kaijun Liu, Shengwei Xu, Guoai Xu, Miao Zhang, Dawei Sun and Haifeng Liu. 2020. A review of android malware detection approaches based on machine learning. *IEEE Access*, 8, 124579–124607. DOI: 10.1109/ACCESS.2020.3006143.

[182] Justin Sahs and Latifur Khan. 2012. A machine learning approach to android malware detection. In *2012 European Intelligence and Security Informatics Conference, EISIC 2012, Odense, Denmark, August 22-24, 2012*. Nasrullah Memon and Daniel Zeng, editors. IEEE Computer Society, 141–147. DOI: 10.1109/EISIC.2012.34.

[183] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon and Konrad Rieck. 2014. DREBIN: effective and explainable detection of android malware in your pocket. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society. https://www.ndss-symposium.org/ndss2014/drebin-effective-and-explainable-detection-android-malware-your-pocket.

[184] Muhammad Ijaz, Muhammad Hanif Durad and Maliha Ismail. 2019. Static and dynamic malware analysis using machine learning. In *16th International Bhurban Conference on Applied Sciences and Technology, IBCAST 2019, Islamabad, Pakistan, January 8-12, 2019*. IEEE, 687–691. DOI: 10.1109/IBCAST.2019.8667136.

[185] Samaneh Mahdavifar, Andi Fitriah Abdul Kadir, Rasool Fatemi, Dima Alhadidi and Ali A. Ghorbani. 2020. Dynamic android malware category classification using semi-supervised deep learning. In *IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress, DASC/PiCom/CBDCom/CyberSciTech 2020, Calgary, AB, Canada, August 17-22, 2020*. IEEE, 515–522. DOI: 10.1109/DASC-PICOM-CBDCOM-CYBERSCITECH49142.2020.00094.

[186] Ahmed Bouteska, Mohammad Zoynul Abedin, Petr Hajek and Kunpeng Yuan. 2024. Cryptocurrency price forecasting – a comparative analysis of ensemble learning and deep learning methods. *Int. Rev. Financ. Anal.*, 92, 103055. DOI: 10.1016/j.irfa.2023.103055.

[187] Patrick Jaquart, Sven Köpke and Christof Weinhardt. 2022. Machine learning for cryptocurrency market prediction and trading. *J. Finance Data Sci.*, 8, 331–352. DOI: 10.1016/j.jfds.2022.12.001.

[188] Pawan Kumar Mall, Pradeep Kumar Singh, Swapnita Srivastav, Vipul Narayan, Marcin Paprzycki, Tatiana Jaworska and Maria Ganzha. 2023. A comprehensive review of deep neural networks for medical image processing: recent developments and future opportunities. *Healthc. Anal.*, 4, 100216. DOI: 10.1016/j.health.2023.100216.

[189] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick and Ali Farhadi. 2016. You only look once: unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 779–788. DOI: 10.1109/CVPR.2016.91.

[190] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov and Liang-Chieh Chen. 2018. Mobilenetv2: inverted residuals and linear bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 4510–4520. DOI: 10.1109/CVPR.2018.00474.

[191] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg and Li Fei-Fei. 2015. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115, 3, 211–252. DOI: 10.1007/S11263-015-0816-Y.

[192] Zulfadzli Drus and Haliyana Khalid. 2019. Sentiment analysis in social media and its application: systematic literature review. *Procedia Comput. Sci.*, 161, 707–714. DOI: 10.1016/j.procs.2019.11.174.

[193] T. K. Shivaprasad and Jyothi Shetty. 2017. Sentiment analysis of product reviews: a review. In *International Conference on Inventive Communication and Computational Technologies, ICICCT 2017, Coimbatore, India, March 10-11, 2017*. IEEE, 298–301. DOI: 10.1109/ICICCT.2017.7975207.

[194] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*. Dekang Lin, Yuji Matsumoto and Rada Mihalcea, editors. The Association for Computer Linguistics, 142–150. https://aclanthology.org/P11-1015/.

[195] VirusTotal. 2025. Virustotal website. Accessed on 2025-09-23. https://www.virustotal.com/.

[196] Contagio Security. 2025. Contagio security blog. Accessed on 2025-09-23. https://contagiodump.blogspot.com/.

[197] George A. Miller. 1994. WORDNET: A lexical database for english. In *Human Language Technology, Proceedings of a Workshop held at Plainsboro, New Jerey, USA, March 8-11, 1994*. Morgan Kaufmann. https://aclanthology.org/H94-1111/.

[198] ImageNet. 2025. Ilsvrc 2012 website. Accessed on 2025-09-23. https://www.image-net.org/challenges/LSVRC/2012/.

[199] Hatoo. 2025. Oha documentation. Accessed on 2025-09-23. https://github.com/hatoo/oha.

[200] Jeffrey Dean and Luiz André Barroso. 2013. The tail at scale. *Commun. ACM*, 56, 2, 74–80. DOI: 10.1145/2408776.2408794.

[201] Kubernetes Developers. 2026. Kubernetes documentation. Accessed on 2026-01-01. https://kubernetes.io/docs/home/.

[202] KServe Developers. 2026. Kserve documentation. Accessed on 2026-01-01. https://kserve.github.io/website/docs/intro.

[203] Kubeflow Developers. 2026. Kubeflow documentation. Accessed on 2026-01-01. https://www.kubeflow.org/docs/.

[204] CoinMarketCap. 2023. Coinmarketcap web service. Accessed on 2025-09-26. https://coinmarketcap.com.

[205] Karl Pearson. 1895. Note on regression and inheritance in the case of two parents. *Proc. R. Soc. Lond.*, 58, 240–242. http://www.jstor.org/stable/115794.

[206] Anoop S Kumar and Taufeeq Ajaz. 2019. Co-movement in crypto-currency markets: evidences from wavelet analysis. *Financial Innovation*, 5, 1, 33. DOI: 10.1186/s40854-019-0143-3.

[207] Harshal Chaudhari and Martin Crane. 2020. Cross-correlation dynamics and community structures of cryptocurrencies. *J. Comput. Sci.*, 44, 101130. DOI: 10.1016/J.JOCS.2020.101130.

[208] Sima Siami-Namini, Neda Tavakoli and Akbar Siami Namin. 2018. A comparison of ARIMA and LSTM in forecasting time series. In *17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018, Orlando, FL, USA, December 17-20, 2018*. M. Arif Wani, Mehmed M. Kantardzic, Moamar Sayed-Mouchaweh, João Gama and Edwin Lughofer, editors. IEEE, 1394–1401. DOI: 10.1109/ICMLA.2018.00227.

[209] George E. P. Box and Gwilym M. Jenkins. 1970. *Time Series Analysis: Forecasting and Control*. Holden-Day.

[210] Syed Yousaf Shah, Dhaval Patel, Long Vu, Xuan-Hong Dang, Bei Chen, Peter Kirchner, Horst Samulowitz, David Wood, Gregory Bramble, Wesley M. Gifford, Giridhar Ganapavarapu, Roman Vaculín and Petros Zerfos. 2021. Autoai-ts: autoai for time series forecasting. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. Guoliang Li, Zhanhuai Li, Stratos Idreos and Divesh Srivastava, editors. ACM, 2584–2596. DOI: 10.1145/3448016.3457557.

[211] Olivier Kraaijeveld and Johannes De Smedt. 2020. The predictive power of public twitter sentiment for forecasting cryptocurrency prices. *J. Int. Financ. Markets Inst. Money*, 65, 101188. DOI: 10.1016/j.intfin.2020.101188.

[212] Clive W. J. Granger. 1969. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, 37, 3, 424–438. http://www.jstor.org/stable/1912791.

[213] Hiro Y. Toda and Taku Yamamoto. 1995. Statistical inference in vector autoregressions with possibly integrated processes. *J. Econometrics*, 66, 1, 225–250. DOI: 10.1016/0304-4076(94)01616-8.

[214] Christopher A. Sims. 1980. Macroeconomics and reality. *Econometrica*, 48, 1, 1–48. http://www.jstor.org/stable/1912017.

[215] Abraham Wald. 1943. Tests of statistical hypotheses concerning several parameters when the number of observations is large. *Trans. Am. Math. Soc.*, 54, 3, 426–482. http://www.jstor.org/stable/1990256.

[216] Denis Kwiatkowski, Peter C. B. Phillips, Peter Schmidt and Yongcheol Shin. 1992. Testing the null hypothesis of stationarity against the alternative of a unit root: how sure are we that economic time series have a unit root? *J. Econometrics*, 54, 1-3, 159–178. DOI: 10.1016/0304-4076(92)90104-Y.

[217] David A. Dickey and Wayne A. Fuller. 1981. Likelihood ratio statistics for autoregressive time series with a unit root. *Econometrica*, 49, 4, 1057–1072. http://www.jstor.org/stable/1912517.

[218] Hirotugu Akaike. 1998. Information theory and an extension of the maximum likelihood principle. In *Selected Papers of Hirotugu Akaike*. Emanuel Parzen, Kunio Tanabe and Genshiro Kitagawa, editors. Springer, 199–213. ISBN: 978-1-4612-1694-0. DOI: 10.1007/978-1-4612-1694-0_15.

[219] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Yoshua Bengio and Yann LeCun, editors. http://arxiv.org/abs/1412.6980.

[220] WalletInvestor. 2025. Walletinvestor website. Accessed on: 2025-09-30. https://walletinvestor.com/.

[221] CryptoPredictions. 2025. Cryptopredictions website. Accessed on: 2025-09-30. https://cryptopredictions.com/.

[222] DigitalCoinPrice. 2025. Digitalcoinprice website. Accessed on: 2025-09-30. https://digitalcoinprice.com/.

[223] NVIDIA. 2025. Cuda toolkit documentation. Accessed on: 2025-09-30. https://docs.nvidia.com/cuda/.

[224] Charles C. Holt. 2004. Forecasting seasonals and trends by exponentially weighted moving averages. *Int. J. Forecast.*, 20, 1, 5–10. DOI: 10.1016/j.ijforecast.2003.09.015.

[225] Peter R. Winters. 1960. Forecasting sales by exponentially weighted moving averages. *Manag. Sci.*, 6, 3, 324–342. http://www.jstor.org/stable/2627346.

[226] Maurice G. Kendall. 1938. A new measure of rank correlation. *Biometrika*, 30, 1-2, 81–93. http://www.jstor.org/stable/2332226.

[227] Charles Spearman. 1904. The proof and measurement of association between two things. *Am. J. Psychol.*, 15, 1, 72–101. http://www.jstor.org/stable/1412159.

[228] Juan José Montaño Moreno, Alfonso Palmer Pol, Albert Sesé Abad and Berta Cajal Blasco. 2013. Using the r-mape index as a resistant measure of forecast accuracy. *Psicothema*, 25, 4, 500–506. DOI: 10.7334/psicothema2013.23.

[229] Etherscan. 2025. Ether total supply and market capitalization chart. Accessed on: 2025-10-01. https://etherscan.io/stat/supply.

[230] Daejun Park, Yi Zhang, Manasvi Saxena, Philip Daian and Grigore Rosu. 2018. A formal verification tool for ethereum VM bytecode. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*. Gary T. Leavens, Alessandro Garcia and Corina S. Pasareanu, editors. ACM, 912–915. DOI: 10.1145/3236024.3264591.

[231] Chainabuse. 2025. Chainabuse. Accessed on: 2025-09-30. https://www.chainabuse.com/.

[232] Jiming Liu and Yiming Ye. 2001. Introduction to e-commerce agents: marketplace solutions, security issues, and supply and demand. In *E-Commerce Agents, Marketplace Solutions, Security Issues, and Supply and Demand* (Lecture Notes in Computer Science). Jiming Liu and Yiming Ye, editors. Vol. 2033. Springer, 1–6. DOI: 10.1007/3-540-45370-9_1.

[233] Jason I. Hong. 2012. The state of phishing attacks. *Commun. ACM*, 55, 1, 74–81. DOI: 10.1145/2063176.2063197.

[234] Cyvers. 2025. Malcon api: advanced malicious contract detection & prevention. Accessed on: 2025-09-30. https://cyvers.ai/malconapi.

[235] Christoph Sendner, Huili Chen, Hossein Fereidooni, Lukas Petzi, Jan König, Jasper Stang, Alexandra Dmitrienko, Ahmad-Reza Sadeghi and Farinaz Koushanfar. 2023. Smarter contracts: detecting vulnerabilities in smart contracts with deep transfer learning. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/smarter-contracts-detecting-vulnerabilities-in-smart-contracts-with-deep-transfer-learning/.

[236] Derek Liu, Francesco Piccoli and Victor Fang. 2023. Machine learning approach to identify malicious smart contract opcodes: a preliminary study. *JPS Conference Proceedings (Blockchain Kaigi 2023)*, 43, 011002. DOI: 10.7566/JPSCP.43.011002.

[237] Tahmina Ehsan, Muhammad Usman Sana, Muhammad Usman Ali, Elizabeth Caro Montero, Eduardo Silva Alvarado, Sirojiddin Djuraev and Imran Ashraf. 2024. Securing smart contracts in fog computing: machine learning-based attack detection for registration and resource access granting. *IEEE Access*, 12, 42802–42815. DOI: 10.1109/ACCESS.2024.3378736.

[238] Jiachi Chen, Xin Xia, David Lo, John Grundy, Xiapu Luo and Ting Chen. 2022. Defectchecker: automated smart contract defect detection by analyzing EVM bytecode. *IEEE Trans. Software Eng.*, 48, 7, 2189–2207. DOI: 10.1109/TSE.2021.3054928.

[239] ConsenSys Diligence. 2025. Mythril: security analysis tool for ethereum smart contracts. Accessed on: 2025-09-30. https://github.com/ConsenSys/mythril.

[240] ETH Zurich SRI Lab. 2025. Securify 2.0. Accessed on: 2025-09-30. https://github.com/eth-sri/securify2.

[241] Josselin Feist, Gustavo Grieco and Alex Groce. 2019. Slither: a static analysis framework for smart contracts. In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB@ICSE 2019, Montreal, QC, Canada, May 27, 2019*. IEEE / ACM, 8–15. DOI: 10.1109/WETSEB.2019.00008.

[242] Bruno Dia, Naghmeh Ramezani Ivaki and Nuno Laranjeiro. 2021. An empirical evaluation of the effectiveness of smart contract verification tools. In *26th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2021, Perth, Australia, December 1-4, 2021*. IEEE, 17–26. DOI: 10.1109/PRDC53464.2021.00013.

[243] Asem Ghaleb and Karthik Pattabiraman. 2020. How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection. In *ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020*. Sarfraz Khurshid and Corina S. Pasareanu, editors. ACM, 415–427. DOI: 10.1145/3395363.3397385.

[244] Gustavo Grieco, Will Song, Artur Cygan, Josselin Feist and Alex Groce. 2020. Echidna: effective, usable, and fast fuzzing for smart contracts. In *ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020*. Sarfraz Khurshid and Corina S. Pasareanu, editors. ACM, 557–560. DOI: 10.1145/3395363.3404366.

[245] Google. 2025. Google bigquery documentation. Accessed on: 2025-09-30. https://cloud.google.com/bigquery/docs.

[246] Etherscan. 2025. Etherscan.io website. Accessed on: 2025-09-30. https://etherscan.io/.

[247] Etherscan Information Center. 2023. Report/flag address. Accessed on 2026-01-01. https://info.etherscan.com/report-address/.

[248] Gibran Gómez, Kevin van Liebergen, Davide Sanvito, Giuseppe Siracusano, Roberto Gonzalez and Juan Caballero. 2024. Sorting out the bad seeds: automatic classification of cryptocurrency abuse reports. *CoRR*, abs/2410.21041. arXiv: 2410.21041. DOI: 10.48550/ARXIV.2410.21041.

[249] Ethereum Improvement Proposals. 2025. Erc-1167: minimal proxy contract. Accessed on: 2025-09-30. https://eips.ethereum.org/EIPS/eip-1167/.

[250] Binance News. 2024. Phishing scam thefts on ethereum layer 2 base surge by 1,900% in q1 2024. Accessed on 2026-01-01. https://www.binance.com/en/square/post/2024-04-02-phishing-scam-thefts-on-ethereum-layer-2-base-surge-by-1-900-in-q1-2024-6223659895817.

[251] Ethereum. 2025. Evmdasm library. Accessed on: 2025-09-30. https://github.com/ethereum/evmdasm.

[252] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang and Ming Zhou. 2020. Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020* (Findings of ACL). Trevor Cohn, Yulan He and Yang Liu, editors. Vol. EMNLP 2020. Association for Computational Linguistics, 1536–1547. DOI: 10.18653/V1/2020.FINDINGS-EMNLP.139.

[253] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu and Shujie Liu. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*. Joaquin Vanschoren and Sai-Kit Yeung, editors. https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c16a5320fa475530d9583c34fd356ef5-Abstract-round1.html.

[254] TonTon Hsien-De Huang and Hung-Yu Kao. 2018. R2-D2: color-inspired convolutional neural network (cnn)-based android malware detections. In *IEEE International Conference on Big Data (IEEE BigData 2018), Seattle, WA, USA, December 10-13, 2018*. Naoki Abe, Huan Liu, Calton Pu, Xiaohua Hu, Nesreen K. Ahmed, Mu Qiao, Yang Song, Donald Kossmann, Bing Liu, Kisung Lee, Jiliang Tang, Jingrui He and Jeffrey S. Saltz, editors. IEEE, 2633–2642. DOI: 10.1109/BIGDATA.2018.8622324.

[255] Hugging Face. 2025. Vit base patch16 224 model card. Accessed on: 2025-09-30. https://huggingface.co/google/vit-base-patch16-224.

[256] Florian Pargent, Florian Pfisterer, Janek Thomas and Bernd Bischl. 2022. Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features. *Comput. Stat.*, 37, 5, 2671–2692. DOI: 10.1007/S00180-022-01207-6.

[257] Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep and Jimmy Lin. 2020. Document ranking with a pretrained sequence-to-sequence model. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020* (Findings of ACL). Trevor Cohn, Yulan He and Yang Liu, editors. Vol. EMNLP 2020. Association for Computational Linguistics, 708–718. DOI: 10.18653/V1/2020.FINDINGS-EMNLP.63.

[258] Athirai A. Irissappane, Hanfei Yu, Yankun Shen, Anubha Agrawal and Gray Stanton. 2020. Leveraging GPT-2 for classifying spam reviews with limited labeled data via adversarial training. *CoRR*, abs/2012.13400. https://arxiv.org/abs/2012.13400 arXiv: 2012.13400.

[259] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*. Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi and George Karypis, editors. ACM, 2623–2631. DOI: 10.1145/3292500.3330701.

[260] Daniele Ucci, Leonardo Aniello and Roberto Baldoni. 2019. Survey of machine learning techniques for malware analysis. *Comput. Secur.*, 81, 123–147. DOI: 10.1016/J.COSE.2018.11.001.

[261] Samuel S. Shapiro and Martin B. Wilk. 1965. An analysis of variance test for normality (complete samples). *Biometrika*, 52, 3-4, 591–611. http://www.jstor.org/stable/2333709.

[262] William H. Kruskal and W. Allen Wallis. 1952. Use of ranks in one-criterion variance analysis. *J. Am. Stat. Assoc.*, 47, 260, 583–621. http://www.jstor.org/stable/2280779.

[263] Sture Holm. 1979. A simple sequentially rejective multiple test procedure. *Scand. J. Stat.*, 6, 2, 65–70. http://www.jstor.org/stable/4615733.

[264] Olive Jean Dunn. 1964. Multiple comparisons using rank sums. *Technometrics*, 6, 3, 241–252. http://www.jstor.org/stable/1266041.

[265] Alexis Dinno. 2015. Nonparametric pairwise multiple comparisons in independent groups using dunn's test. *Stata J.*, 15, 1, 292–300. DOI: 10.1177/1536867X1501500117.

[266] Janez Demsar. 2006. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7, 1–30. https://jmlr.org/papers/v7/demsar06a.html.

[267] Milton Friedman. 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *J. Am. Stat. Assoc.*, 32, 200, 675–701. http://www.jstor.org/stable/2279372.

[268] Frank Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometrics Bull.*, 1, 6, 80–83. http://www.jstor.org/stable/3001968.

[269] Norman Cliff. 1993. Dominance statistics: ordinal analyses to answer ordinal questions. *Psychol. Bull.*, 114, 3, 494–509. DOI: 10.1037/0033-2909.114.3.494.

[270] William Jay Conover. 1999. *Practical Nonparametric Statistics*. (3rd ed.). John Wiley & Sons, New York, NY, USA.

[271] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder and Lorenzo Cavallaro. 2019. TESSERACT: eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*. Nadia Heninger and Patrick Traynor, editors. USENIX Association, 729–746. https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury.

[272] Scott M. Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan and Roman Garnett, editors, 4765–4774. https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html.

[273] Scikit-learn developers. 2025. Permutation feature importance. Accessed on: 2025-09-30. https://scikit-learn.org/stable/auto_examples/inspection/plot_permutation_importance.html.

[274] Etherscan. 2025. Smart contract 0xb5e7b87e7a84276b13da3f07495e18f3e229d3a0. Accessed on: 2025-09-30. https://etherscan.io/address/0xb5e7b87e7a84276b13da3f07495e18f3e229d3a0.

[275] Etherscan. 2025. Smart contract 0x279e2f385ce22f88650632d04260382bfb918082. Accessed on: 2025-09-30. https://etherscan.io/address/0x279e2f385cE22F88650632D04260382bFB918082.

[276] Mikhail Belkin, Daniel Hsu, Siyuan Ma and Soumik Mandal. 2018. Reconciling modern machine learning and the bias-variance trade-off. *CoRR*, abs/1812.11118. http://arxiv.org/abs/1812.11118 arXiv: 1812.11118.

[277] Cointelegraph. 2022. Web3 sees 15 new scam smart contracts an hour: solidus labs. Accessed on 2025-11-26. https://cointelegraph.com/news/web3-sees-15-new-scam-smart-contracts-an-hour-solidus-labs.

[278] Polygon Labs. 2025. Polygon: scalable, interoperable, multi-chain protocols. Accessed on 2025-11-26. https://polygon.technology/.

[279] BNB Chain. 2025. Bnb chain: build natively on the world's largest smart contract network. Accessed on 2025-11-26. https://www.bnbchain.org/en.

[280] Jonathan Becker. 2025. Heimdall-rs: an advanced evm smart contract toolkit. Accessed on 2025-11-26. https://github.com/Jon-Becker/heimdall-rs.

[281] Vincenzo Arceri, Saverio Mattia Merenda, Luca Negrini, Luca Olivieri and Enea Zaffanella. 2025. EVMLiSA: sound static control-flow graph construction for EVM bytecode. *Blockchain: Research and Applications*, 100384. DOI: 10.1016/j.bcra.2025.100384.

[282] Dxo, Mate Soos, Zoe Paraskevopoulou, Martin Lundfall and Mikael Brockman. 2024. Hevm, a fast symbolic execution framework for EVM bytecode. In *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I* (Lecture Notes in Computer Science). Arie Gurfinkel and Vijay Ganesh, editors. Vol. 14681. Springer, 453–465. DOI: 10.1007/978-3-031-65627-9_22.

[283] Etherscan. 2025. How to verify contracts on etherscan. Accessed on 2025-11-26. https://info.etherscan.com/how-to-verify-contracts/.

[284] Pietro Ferrara, Luca Negrini, Vincenzo Arceri and Agostino Cortesi. 2021. Static analysis for dummies: experiencing lisa. In *SOAP@PLDI 2021: Proceedings of the 10th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis, Virtual Event, Canada, 22 June, 2021*. Lisa Nguyen Quang Do and Caterina Urban, editors. ACM, 1–6. DOI: 10.1145/3460946.3464316.

[285] Alberto Sanfeliu and King-Sun Fu. 1983. A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. Syst. Man Cybern.*, SMC-13, 3, 353–362. DOI: 10.1109/TSMC.1983.6313167.

[286] Harold W. Kuhn. 1955. The hungarian method for the assignment problem. *Naval Res. Logist. Q.*, 2, 1-2, 83–97. DOI: 10.1002/nav.3800020109.

[287] Patrick P. F. Chan and Christian Collberg. 2014. A method to evaluate CFG comparison algorithms. In *2014 14th International Conference on Quality Software (QSIC)*, 95–104. DOI: 10.1109/QSIC.2014.28.

[288] Aina Niemetz and Mathias Preiner. 2023. Bitwuzla. In *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II* (Lecture Notes in Computer Science). Constantin Enea and Akash Lal, editors. Vol. 13965. Springer, 3–17. DOI: 10.1007/978-3-031-37703-7_1.

[289] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva and Antonio Torralba. 2016. Learning deep features for discriminative localization. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2921–2929. DOI: 10.1109/CVPR.2016.319.

[290] Pasquale De Rosa, Pascal Felber and Valerio Schiavoni. 2025. Scamdetect: towards a robust, agnostic framework to uncover threats in smart contracts. In *2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2025 - Supplemental Volume, Naples, Italy, June 23-26, 2025*. IEEE, 242–244. DOI: 10.1109/DSN-S65789.2025.00068.

[291] Roy Frostig, Matthew Johnson and Chris Leary. 2018. Compiling machine learning programs via high-level tracing. In *Proceedings of Machine Learning and Systems 2018, MLSys 2018*. https://mlsys.org/Conferences/doc/2018/146.pdf.

[292] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama and Trevor Darrell. 2014. Caffe: convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia, MM '14, Orlando, FL, USA, November 03 - 07, 2014*. Kien A. Hua, Yong Rui, Ralf Steinmetz, Alan Hanjalic, Apostol Natsev and Wenwu Zhu, editors. ACM, 675–678. DOI: 10.1145/2647868.2654889.