PhishingHook: Catching Phishing Ethereum Smart Contracts leveraging EVM Opcodes

Pasquale De Rosa [†], Simon Queyrut [†], Yérom-David Bromberg [*], Pascal Felber [†], and Valerio Schiavoni [†]
[*]University of Rennes, CNRS, INRIA, IRISA, Rennes, France, david.bromberg@irisa.fr
[†]University of Neuchâtel, Neuchâtel, Switzerland, first.last@unine.ch

***Abstract*—The Ethereum Virtual Machine (EVM) is a decentralized computing engine. It enables the Ethereum blockchain to execute smart contracts and decentralized applications (dApps). The increasing adoption of Ethereum sparked the rise of phishing activities. Phishing attacks often target users through deceptive means, *e.g.*, fake websites, wallet scams, or malicious smart contracts, aiming to steal sensitive information or funds. A timely detection of phishing activities in the EVM is therefore crucial to preserve the user trust and network integrity. Some state-of-the art approaches to phishing detection in smart contracts rely on the online analysis of transactions and their traces. However, replaying transactions often exposes sensitive user data and interactions, with several security concerns. In this work, we present PHISHINGHOOK, a framework that applies machine learning techniques to detect phishing activities in smart contracts by directly analyzing the contract's bytecode and its constituent opcodes. We evaluate the efficacy of such techniques in identifying malicious patterns, suspicious function calls, or anomalous behaviors within the contract's code itself before it is deployed or interacted with. We experimentally compare 16 techniques, belonging to four main categories (Histogram Similarity Classifiers, Vision Models, Language Models and Vulnerability Detection Models), using 7,000 real-world malware smart contracts. Our results demonstrate the efficiency of PHISHINGHOOK in performing phishing classification systems, with about 90% average accuracy among all the models. We support experimental reproducibility, and we release our code and datasets to the research community.**

***Index Terms*—EVM, opcodes, smart contracts, phishing, detection**

I. INTRODUCTION

Blockchains are distributed ledgers used across several application domains [7], [12], [18]. They offer well-desired properties, such as anti-tampering, scalability, *etc*. The Ethereum blockchain [84] moves nowadays large financial assets and supports an increasing number of financial transactions. With a market cap of several billion USD [28], Ethereum is the 2nd most used blockchain. Ethereum natively supports sophisticated decentralized applications (*i.e.*, *dApps*) implemented by means of special blockchain programs, *i.e.*, smart contracts. The execution of smart contracts operates via the Ethereum Virtual Machine (EVM), a formally-verifiable [61] decentralized computing engine. In a nutshell, the EVM is a stack-based machine that executes *opcodes*, translated to primitive low-level operations such as arithmetic, memory, or stack manipulations (see §II).

The Ethereum blockchain is a frequent target of cybercriminal attacks, extensively studied in the literature [6], [10], [49], [63]. Several categories of attack exist: reentrancy attacks (*e.g.*, the famous DAO attack [74] or [47], leading to dozens of lost millions), critical reentrancy attacks, default visibility weaknesses, arithmetic under/over flows due to the internal representation of integer values in the EVM, use of custom and malformed random functions [65], front running attacks [80], recent typosquatting attacks [69], and more. Among these, phishing attacks stand out as one of the most prevalent threats in malicious smart contracts and the main concern for the Ethereum community, as reported by ChainAbuse [9]. While website phishing attacks have long been a concern [42], [55], their rapid growth on popular blockchains has recently attracted significant attention from researchers and practitioners.

We focus on a specific class of phishing attacks, where early detection is crucial to mitigate (or ideally prevent) damage. While commercial services for detecting malicious smart contracts are readily accessible to users [17], they are often costly. Although various detection techniques exist (see later in §VI), there is a lack of open-source and reproducible experimental comparisons of their accuracy.

To address this gap, we present PHISHINGHOOK, the first framework designed to easily evaluate and compare various techniques for detecting phishing smart contracts, the most predominant type of malicious smart contracts. Specifically, we evaluate 16 machine learning (ML) models, including seven – ViT+R2D2, ViT+Freq, ESCORT, GPT-2 and T5 (in two variants each) – that have not been previously tested in the context of opcode-based phishing detection for smart contracts. The remaining nine models, while having been assessed on related datasets (such as fraud detection containing phishing samples), are evaluated for the first time on a dedicated phishing dataset.

To the best of our knowledge, PHISHINGHOOK is the only framework focused exclusively on phishing detection on smart contracts using opcode analysis. Its architecture, detailed in §III includes components to extract the bytecode of a smart contract, a bytecode disassembler, a model evaluation module, and finally a post hoc analysis. Additionally, PHISHINGHOOK provides tools to build or extend existing datasets by crawling public Ethereum explorers or using query services. The contributions of this paper are:

- The construction and public release of the largest dataset of phishing smart contracts targetting the Ethereum blockchain, available at [70];

TABLE I: EVM opcodes for the Shanghai fork (from [30]).

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

- An architecture, prototype implementation and experimental evaluation of PHISHINGHOOK;
- The analysis of the accuracy of 16 different detection techniques, including statistical approaches, machine-learning methods, computer vision approaches, LLM and vulnerability detectors.
- The full set of instructions to reproduce our experiments.

Roadmap. We overview Ethereum and the required EVM internals in §II. The architecture of PHISHINGHOOK is described in §III. We report on our experimental evaluation in §IV. We detail the lessons learned in §V. We survey related work in §VI, before concluding in §VII.

II. BACKGROUND

Ethereum and smart contracts. Ethereum [84] is a state machine where transactions trigger valid state transitions. These transactions are bundled into blocks, cryptographically linked to form a chain. Each block acts as a journal, recording its transactions, the hash of the previous block, and the resulting state. State changes are executed by the Ethereum Virtual Machine (EVM), governed by a *gas* parameter that bounds computation. Transactions come in two forms: *(i)* message calls and *(ii)* contract creation, which deploys accounts with code (*i.e.*, smart contracts). Smart contracts are on-chain programs that algorithmically enforce agreements, combining persistent storage with executable functions, typically written in Solidity [75] or Viper [81]. Ether (ETH) is Ethereum's native currency. Consensus on new blocks is achieved through the Beacon Chain using proof-of-stake (PoS). This work focuses on Ethereum starting from the Shanghai update at block 17034870.

Ethereum Virtual Machine (EVM). The EVM [84] is a 256-bit stack machine with a maximum of 1024 stack items. Both memory and storage are word-addressed byte arrays, initialized to zero; storage is non-volatile and part of the global state. Execution halts on stack underflows, invalid instructions, or gas exhaustion. Execution is opcode-driven: each opcode represents a specific operation such as arithmetic (ADD, SUB), signed math (SDIV, SMOD), hashing (SHA3), memory/stack manipulation, or contract execution (CALL, DELEGATECALL, etc.). Contracts can be created (CREATE, CREATE2) or removed (SELFDESTRUCT) via dedicated opcodes. As of the Shanghai update, 144 opcodes exist. Table I lists several, with a complete reference in [30].

Phishing attacks in the Ethereum blockchain. Phishing attacks in Ethereum involve tricking users into approving harmful transactions or exposing private keys through impersonation of trusted platforms (e.g., dApps [78]). Attackers bait victims with fake incentives (airdrops, staking) via social media or emails, directing them to fraudulent sites mimicking legitimate dApps. After wallet connection, victims are prompted to approve a transaction (e.g., "claim reward"), which secretly authorizes attackers to drain their funds.

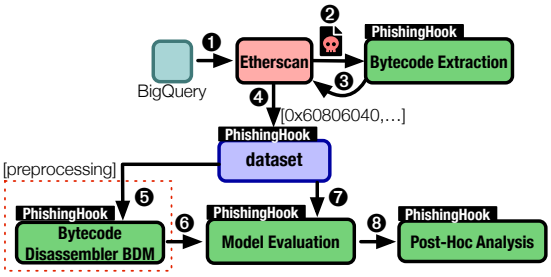

Fig. 1: The PHISHINGHOOK framework.

III. THE PHISHINGHOOK FRAMEWORK

The architecture of PHISHINGHOOK consists of four core modules: *(i)* bytecode extraction module (BEM), *(ii)* bytecode disassembler module (BDM), *(iii)* model evaluation module (MEM), and *(iv)* a post hoc analysis module (PAM). An initial data gathering phase leverages etherscan.io [29] and Google BigQuery [38]. We describe the details below.

Data gathering. We first gather a raw list of unlabeled contract hashes from the Ethereum public dataset available on Google BigQuery (Fig. 1-❶). For this study, we limited our search to the contracts deployed between October 2023 and October 2024 ($\approx 4,000,000$). As of October 22, 2024, the total number of contracts deployed on the Ethereum blockchain and available on the BigQuery dataset was 68,681,183. The public service etherscan.io flags phishing smart contracts with the label "Phish/Hack" (Fig. 1-❷). We leverage this service to scrape data for each of the 4 million hashes. Etherscan acts as an independent source of smart contract validation and security analysis. Other sources based on community reports (ChainAbuse) are currently proven to be biased [37].

Bytecode extraction module (BEM). The first step of PHISHINGHOOK consists in the extraction of the bytecode from the retrieved and labeled contracts. To do so, we rely on a public etherscan endpoint (`eth_getCode`) *via* an JSON-RPC API (Fig. 1-❸). The resulting extracted bytecode constitutes the core dataset adopted in the model training and evaluation phase (Fig. 1-❹).

Dataset construction. From the approximately 4 million contracts collected in the data gathering phase, we sample 17,455 phishing bytecodes, with 3,458 unique bytecodes (Fig. 2). We notice indeed a large majority of duplicate (bit-by-bit) bytecodes. The reason for such duplicates is the presence of a significant amount of minimal proxy contracts [64], *i.e.*, lightweight and cost-efficient "clones" of a main contract, with which they share the same bytecode. In addition to the malicious smart contracts, we enrich the dataset by a similar number of benign samples (*i.e.*, in our context, not "flagged" as malicious on etherscan.io), constituting a final dataset of

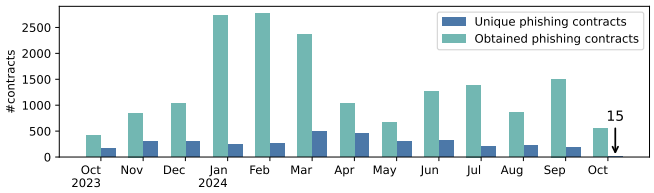

Fig. 2: Number of phishing contracts per month over the 2023-10 to 2024-10 period.

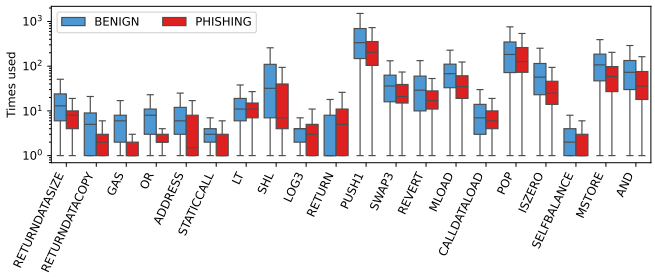

Fig. 3: Distribution, by opcode usage, of contracts for 20 opcodes.

7,000 bytecodes. We release this novel dataset via our public repository: [70]. To our knowledge, this is the largest dataset of phishing smart contracts available to the research community.

Bytecode disassembler module (BDM). The deployed bytecode of a smart contract is always publicly accessible on the blockchain, whereas the source code and its AST must be explicitly disclosed by the contract creators. Since malicious actors may choose to withhold or obfuscate source code containing harmful functions, we focus on analyzing the deployed bytecode. The BDM module is in charge of disassembling the bytecode into its corresponding series of opcode instructions (Fig. 1-❺), *i.e.*, mnemonic (human-readable alias), operand (argument) and gas (execution cost). BDM is a required step only for some of the evaluated detection models (*i.e.*, Histogram Similarity Classifiers and ViT + Frequency), that rely on disassembled bytecode features and cannot be trained directly on the original binary. For example, a simple bytecode `0x6080604052` gets disassembled to: `(PUSH1, 0x80, 3), (PUSH1, 0x40, 3), (MSTORE, NaN, 3)`. In PHISHINGHOOK, the resulting disassembled opcodes are stored in a `.csv` file for further processing (Fig. 1-❻). To perform the disassembling, we leverage a modified version of the Python library evmdasm [27], that we enhanced in order to handle the EVM opcode instructions as of the Shanghai fork. The last version of evmdasm registry has been released in March 2022, during the Arrow Glacier update. We added support for two new opcodes: INVALID (that designates an invalid instruction) and PUSH0 (to push 0 bytes in stack). We release this enhanced version of evmdasm via our public repository: [70]. One might question whether a discernible pattern exists that could help differentiate phishing contracts from non-phishing ones, based on the relative prevalence of certain opcodes. We show the distribution of contracts based on how frequently they use each of 20 influential (see §IV-H) opcodes (Fig. 3). Phishing contracts utilize opcodes at a similar rate as their benign counterparts, making it unreliable to filter samples solely based on the frequency of a single opcode.

Model evaluation module (MEM). This module comprises the systematic training and evaluation of ML models to classify phishing smart contracts from their bytecode representation (Fig. 1-❼). We consider 16 different models from 4 different categories: (i) histogram similarity classifiers (HSCs), (ii) vision models (VMs), (iii) language models (LMs) and (iv) vulnerability detection models (VDMs). We compare state-of-the-art, industry-proven models also explored in similar domains, *e.g.*, malware detection in smart contracts. These include HSCs, ECA+EfficientNet, and SCSGuard, as well as other recent models from different domains, such as ViT, GPT-2, and T5. We provide more details on the models and our implementation in §IV-A and §IV-B, respectively. We opted for transformers over traditional sequence models (RNNs) for their superior performance in representation learning for code sequences and detection [34], [56].

Post hoc analysis module (PAM). Finally, we conduct a post hoc analysis on the performances of the ML models (Fig. 1-❽). The scope of this analysis is to determine that significant differences exist among the performance metrics of the evaluated models. The adoption of the PAM in PHISHINGHOOK ensures the statistical validity and significance of the observed results in the evaluation phase. Our scripts for the post hoc analysis are written in R. We provide additional details on PAM in §IV-E.

IV. EVALUATION

A. Experimental setting

We use two different GPU-enabled nodes. The first features an Intel Core i7-14700KF (28 cores, 5.5 GHz), 64 GiB of RAM, and an NVIDIA GeForce RTX 4090 GPU with 24 GiB of VRAM, utilizing CUDA 12.2. The second is powered by an Intel Xeon Platinum 8562Y+ (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 deep-learning based models, the design of the train-test pipeline leverages Pytorch v2.5. For classical ML models, we use SciKit-learn v1.5. The post hoc analysis module uses R v4.4.

B. Compared models

PHISHINGHOOK solves a binary classification task, *i.e.*, determining whether the sample is a phishing contract or not. We next describe the feature extraction methods and models incorporated in PHISHINGHOOK. As the original authors did not release their implementations as open source, we reimplemented all of them from scratch within the PHISHINGHOOK framework. We include in our experimental study several static code analysis protocols. We consider 16 methods, detailed next.

HSC (Histogram Similarity Classifiers) [54]: For each contract bytecode, a histogram of the occurrences of opcodes is created. It builds a vector of length equal to the number of unique opcodes inside the training set. The vector is directly

served as input (*i.e.*, without normalized nor standardized steps) to several classical ML classifiers proposed by scikit-learn: Random Forest, LightGBM, kNN, XGBoost, CatBoost, Logistic Regression, SVM [71].

ViT+R2D2 (Vision Model): Building on methods from Android malware detection research (R2D2 [44]), we interpret the bytecode as a sequence of hexadecimal color codes. Each hexadecimal value in the bytecode is mapped to a color in the RGB space. All pixels (*i.e.*, three channels of integers) are arranged into a $224 \times 224 \times 3$ tensor, with zero-padding applied as needed. The resulting tensor will serve as input to a classifier. Due to their widely recognized performance in image classification tasks, we selected the Vision Transformer [23] to predict the class of the associated contract. In this study, we utilize a ViT-B/16 model pretrained on ImageNet-1k, with weights obtained from Hugging Face [31] and fine-tuned on our binary classification task.

ECA+EfficientNet [85] (Vision Model): The bytecodes of Ethereum smart contracts are transformed into RGB image representations, following a process analogous to ViT+R2D2. The feature extractor incorporates a modified ECA module [82], which enhances the model's ability to focus on essential channels. The ECA mechanism computes a channel-wise attention vector using a depthwise convolution. The resulting vector is fed to the backbone of the classifier, which leverages a modified EfficientNet-B0 architecture [77]. Global average pooling is applied to reduce spatial dimensions, followed by a fully connected layer for classification.

ViT+Freq (Vision Model): A lookup table encodes each opcode and operand of the disassembled bytecode to a numerical value which corresponds to their frequency of appearance in the training set. This frequency is mapped into a color channel intensity value. The lookup table is constructed exactly once on the entire contract training set. This frequency encoding performs efficiently as a categorical encoding technique [60]. The concept relies on assigning higher pixel intensity values in the R, G, and B channels to the most frequently encountered mnemonics, operands and gas consumptions. The image tensors have a fixed size of $224 \times 224 \times 3$ which are accordingly zero-padded and subsequently fed to a pretrained ViT-B/16 model (*e.g.*, same as in ViT+R2D2).

SCSGuard [43] (Language Model): Each hexadecimal string within the bytecode is read as a bigram (sequences of 6 characters). These bigrams are numerically encoded to create a vocabulary (*i.e.*, a list of integers), and the sequences are padded to uniform lengths to enable processing by the model. SCSGuard begins with an embedding layer that maps bigram indices to dense vectors. A multi-head attention mechanism is applied to capture dependencies between different parts of the sequence, followed by a GRU layer that models sequential patterns in the data. Finally, a fully connected linear layer generates the logits, representing the model's predictions.

GPT-2 [67] and T5 [68] (Language Models): GPT-2 and T5 are two Transformer-based language models pretrained on a large corpus of text. We selected the latest versions of the models that we could retrieve from the HuggingFace library [45]. Despite their modest size (up to 1.5 billion parameters), these models can be leveraged to build good classifiers [48], [58]. Since transformer-based models interpret text as a sequence of tokens, we use the suitable `GPT2Tokenizer` and `T5Tokenizer` classes from Hugging Face to process the textual inputs.

ESCORT [72] (Vulnerability Detection Model): This system is originally designed to detect code vulnerabilities in smart contracts, and is hereby used for the first time in the context of fraud detection. ESCORT embeds the smart contract bytecode into a vector space. The generated feature representations are then processed by a deep neural network (DNN) model for further analysis. The design supports two operational modes: an initial training phase, where the model is trained to classify individual vulnerabilities in a multi-class labeling context, and a second phase, where the aim is to detect new vulnerabilities through transfer learning.

C. Hyperparameter search

We rely on Optuna [4], an open-source hyperparameter optimization framework designed to automate and streamline the tuning process in ML workflows, to select hyperparameters for all models. Optuna uses metaheuristics to find the best hyperparameters for models by implementing a define-by-run API, which allows users to dynamically construct search spaces. We conducted grid search over an arbitrary search space on the same task as the main evaluation, using 10-fold cross-validation to ensure robust performance assessment.

D. Results

To ensure stability of the results, we performed a 10-fold cross-validation over 3 runs, for a total of 30 experiments per model (not including the hyperparameter runs). Table II shows the results, averaged over 10 folds and 3 runs. We evaluate two variants of GPT-2 and T5: α, where opcode sequences are truncated to fit model token limits and GPU constraints (RTX 4090), and β, trained on an H100 NVL, where full bytecodes are processed in chunks using a sliding window. SCSGuard, relying on n-grams, remains unaffected. Our results indicate that the vulnerability detector ESCORT is not effective when adapted to a different task like the classification of phishing smart contracts, mostly due to inner limitations of VDMs when applied to social engineering vulnerabilities like phishing, since they exploit human behavior rather than technical flaws in the code.

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%. In particular, the Random Forest approach is both the best performing model overall and the best performing HSC. The LMs ranked 2$^{\text{nd}}$, with an average Accuracy of 88.83%, F1 Score of 88.17%, Precision of 89.50% and Recall of 88.07%. SCSGuard is the best performing LM model. The VMs reported an average Accuracy of 83.75%, F1 Score of 83.40%, Precision of 83.26% and Recall of 83.63%. ECA+EfficientNet is the best performing vision model. All models achieve reasonable performance in detecting phishing

TABLE II: Averaged performance metrics for the models supported in PHISHINGHOOK (best values in bold). †: 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

from bytecode, with an average Accuracy of 89.07%, F1 Score of 88.74%, Precision of 89.24% and Recall of 88.70%.

Our results align with the original papers. For HSCs [54], Random Forest was also the best model but had slightly lower accuracy (85.17%). In contrast, ECA+EfficientNet [85] achieved 98.2%. However, these evaluations were conducted on broader fraud datasets, making direct comparison with our results, focused solely on phishing, challenging. SCSGuard [43] detected four out of five phishing scams, with a lower accuracy (80%). Other models were our own implementations; ESCORT, designed for vulnerability detection, had not been used for fraud detection. The discussed metrics are crucial for demonstrating the performance of a malware analysis system [79]. However, the trade-offs with cost metrics, such as training/inference time in relation to data size, must also be considered; they're addressed in greater detail in §IV-F.

E. Post hoc analysis

In this section, we describe the methodologies applied for the post hoc analysis of results. Specifically, our aim is to rigorously compare the performance metrics (Accuracy, Precision, Recall, and F1 Score) obtained for each model to assess the significance of the observed differences.

We excluded from the post hoc analysis the vulnerability detector ESCORT, as it did not perform well on the phishing detection task. We also excluded GPT-2_β and $T5_\beta$, as they were the worst-performing variants of these models. We conducted the post-hoc analysis on the full experimental results - namely, 30 trials per model (10-fold cross validation $\times$ 3 runs). As a result, the number of observation for each metric was 390 (13 models $\times$ 30 trials). Initially, we tested the normality of each metrics distribution with the Shapiro-Wilk (S-W) test [73]. This represents a crucial step: the following choices of which test to adopt for the group comparison (parametric *vs.* non-parametric) are based on this assumption. As of the S-W, the test statistic W (a numerical value that indicates the likelihood of observing the results under the null hypothesis) is computed as $\sum_{i=1}^{n} a_i x_{(i)}^2 / \sum_{i=1}^{n}(x_i - \bar{x})^2$, where $x_{(i)}$ are the ordered sample observations, a_i are the coefficients, $\bar{x}$ is the sample mean, and n the sample size.

TABLE III: Results of performance metrics with the Kruskal-Wallis test. 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}

The null hypothesis of normality is rejected for values of W significantly lower than 1, such that $p < 0.05$. In our case, the assumption of normality was violated for 20 model-metric pairs (out of 52). Hence, we opted for the non-parametric Kruskal-Wallis (K-W) test [51]. K-W is used to determine whether there are statistically significant differences between the medians of three or more independent groups (in our case, the performance metrics for each of the 13 models).

The K-W test statistic, conventionally named H, is computed as $12/(N(N+1)) \cdot \sum_{i=1}^{k} R_i^2/n_i - 3(N+1)$, where k are the number of groups, n_i the number of observations for the i^{th} group, N the total number of observations (for all groups combined) and R_i the sum of ranks, which represent the positions of data points in the i^{th} group.

The null hypothesis that all the performance metrics have the same median is rejected for values of H significantly high, such that $p < 0.05$. In this test, we adjust p to p_{adj} with the Holm-Bonferroni correction to reduce the risk of false positives [3]. As shown in Table III, the null hypothesis is firmly rejected for all four metrics, thus proving the existence of significant differences between the model performances. For this reason, we completed our post hoc analysis performing a Dunn's test [24] with the Holm-Bonferroni correction to determine exactly which model pairs diverge, for any of the four metrics. This is the appropriate nonparametric pairwise multiple comparison procedure when a Kruskal–Wallis test is rejected [22]. The Dunn's test statistic, conventionally named Z, is computed as $(\bar{R}_i - \bar{R}_j)/(\sqrt{(N(N+1))/12) \cdot (1/n_i + 1/n_j)}$, where $(\bar{R}_i - \bar{R}_j)$ is the difference in mean ranks between the two group pairs i and j, and the denominator is the variance term that accounts for the variability in ranks, adjusted for sample sizes of the pairs (n_i, n_j).

Fig. 4 reports these results. For Accuracy, F1 Score and Precision, the percentage of model pairs that showed significant differences in performance was 65.38%, while for Recall it was 61.54%. For model pairs belonging to the same category (*e.g.*, SVM and k-NN are both HSC while GPT-2 and T5 are both LMs) the percentage of significant differences was lower, *i.e.*, 37.04% for Accuracy and F1 Score, 40.74% for Precision and 33.33% for Recall. Instead, for model pairs belonging to different categories, this percentage increased substantially, *i.e.*, 80.39% for Accuracy and F1 Score, 78.43% for Precision and 76.47% for Recall. On average, the observed results are extremely significant for model pairs not belonging to the same category, while for similar models there was less statistical relevance in the observed divergencies.

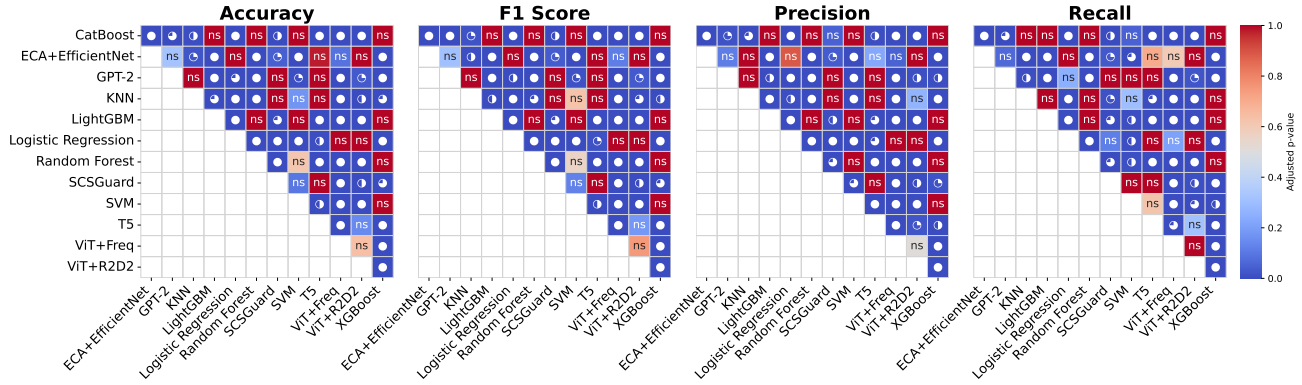

Fig. 4: Dunn's test for pairwise comparison between each model's metrics. Significant if $p_{\text{adj}} < 0.05$. Significance levels from ● (highly significant) down to ◔ (mildly significant). Non-significant results are labeled as **ns**.

F. Model scalability analysis

Here we conduct a model scalability evaluation. The goal is to study the impact of data size on the performances of the considered models. We generated three data splits, containing respectively 1/3, 2/3 and all the smart contract samples. We trained and evaluated three models (SCSGuard, ECA+EfficientNet, Random Forest), *e.g.*, the most accurate models in each category, respectively LM, VM and HSC. Fig. 5 reports our results. Random Forest is the most accurate model for each of the data splits and its performances remain stable. SCSGuard and ECA+EfficientNet scale better when increasing the number of training samples. This suggests that by increasing the available contract bytecodes in the dataset, Vision and Language models will outperform the HSCs.

Moreover, we adopt the Critical Difference Diagram (CDD) [19] to concisely represent a post hoc analysis on the observed results, shown in Fig. 6. To produce the CDD, a Friedman test [35] is first conducted to detect whether there are significant differences in the observed model metrics among the data splits. If some differences are detected, the second step is to perform a Wilcoxon signed-rank test [83] to identify which pair of models exhibit a significant difference. The rightmost elements in the CDD are the ones that showed the best performances (Random Forest for all the four metrics). The leftmost ones are the least performing models (ECA+EfficientNet). The thick horizontal line connects the three classifiers that failed the Wilcoxon test, indicating no statistical evidence. Across all comparisons, p is 0.25 or 0.75, with $p_{adj} = 0.75$. Cliff's δ [14] suggests 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), reflecting performance declines. However, the high p_{adj} values indicate that these differences are not statistically significant. This is likely due to the small sample size in the scalability experiment (36 measurements in total), as non-parametric methods require larger datasets for robust statistical power [15].

We further study the training and inference times for such models, shown in Fig. 7. The impact of data augmentation on complex models, in particular for Language models, is

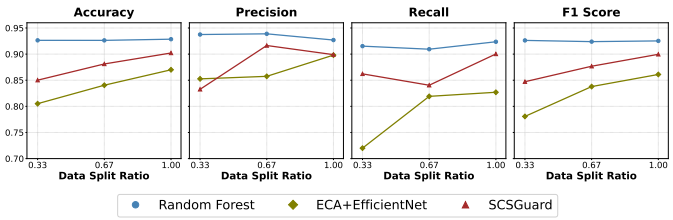

Fig. 5: Performance metrics of the best models per data split.

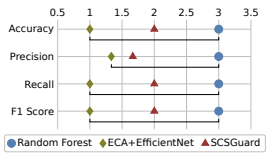

Fig. 6: Critical difference diagram of model scalability.

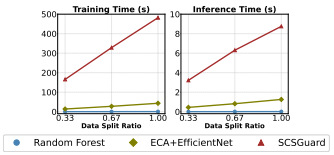

Fig. 7: Time metrics of the best models per data split.

extremely costly. On average (among all data splits), SCSGuard has a training time of 325.302 seconds (+64733.4% respect to the Random Forest and +1030.6% respect to ECA+EfficientNet). The average inference time on one batch of data is 6.091 seconds (+57258.5% respect to the Random Forest and +622.4% respect to ECA+EfficientNet). As we can see, for SCSGuard, both training and inference times almost double (+77.35% on average training time and +68.8% on average inference time) each time the data sample is enlarged, while for both HSCs and the VM they remain stable and low. Therefore, incrementing the number of contract samples in the dataset can increase significantly the performances of complex models, and especially the Language ones. However, at the same time, the associated loss in time efficiency should be considered. The importance of timeliness is generally application-specific: in crypto wallets, users interact with smart contracts in real-time, often signing transactions within seconds. Any delay in detecting a phishing contract could mean a user already approved a malicious transaction before getting a warning. DeFi projects list new tokens frequently, and malicious projects can rug-pull within minutes while security firms analyze threats over time, so delays aren't as critical.

G. Time-resistance analysis

Building on [62], we design a time-resistance experiment to evaluate PHISHINGHOOK's robustness against temporal performance decay. We construct a second dataset of 7,000 samples, ensuring benign samples match the temporal distribution of phishing ones (Fig. 2). The training set includes smart contracts from October 2023 to January 2024, while nine test sets, spanning February to October 2024, assess performance over time. As in the scalability experiment, we evaluate SCSGuard, ECA+EfficientNet, and Random Forest to determine whether phishing contracts exhibit persistent patterns or evolve to evade detection. Results (Fig. 8) show stable phishing detection performance, with only a slight decline due to evolving attack patterns, as explained in the reference study. To quantify stability, we use Area Under Time (AUT), with AUT $\in [0, 1]$, which represents the area under the F1 curve for phishing samples. A higher AUT indicates greater robustness against evolving threats. Random Forest achieves the highest stability (AUT = 0.89), followed by SCSGuard (0.84) and ECA+EfficientNet (0.79), which experiences more fluctuations. Despite minor variations, PHISHINGHOOK effectively adapts to new threats, proving to be a reliable long-term phishing detection solution. The dataset for this analysis is publicly available at [70].

H. Influence of opcode prevalence on the best classifier

Beyond the statistical analysis presented earlier, examining the influence scores of the opcodes provides complementary insights for the model designer. Due to space concerns, we restrict ourselves to the best-performing model, HSC with Random Forests. Fig. 9 displays the Shapley values [57] associated with the 700 samples that comprise the test set of a random fold from §IV-D (other interpretability tools exist, such as impurity-based feature importance, but they can be biased towards high-cardinality features which can result in misleading importance scores [52]). The vertical axis represents the SHAP value, which quantifies the contribution of each feature (here, opcode usage) along the horizontal axis in shifting the model's prediction for that instance away from the base value (*i.e.*, the mean probability of phishing across all contracts). For instance, the cluster of contracts with Shapley values of 0.025 that use GAS very rarely (3rd column) suggests the classifier finds low usage of GAS suspicious. Many well-structured contracts manage gas efficiently, especially when dealing with external calls (*e.g.*, as in `Address.functionDelegateCall(address(this), data[i])` [2]) for which controlled execution may explicitly check the available gas before proceeding. Phishing contracts, on the other hand, often lack such safety checks because they are designed to steal funds quickly rather than execute complex operations. However, operations that trigger GAS can be nested inside loops (*e.g.*, a source code can contain `if (deprecated) {return UpgradedStandardToken(upgradedAddress).transferByLegacy(msg.sender, _to, _value);}` [1]) and still be present in the disassembled bytecode, which can dilute the behavior and render this indicator unreliable (in this case, the contract was a false negative).

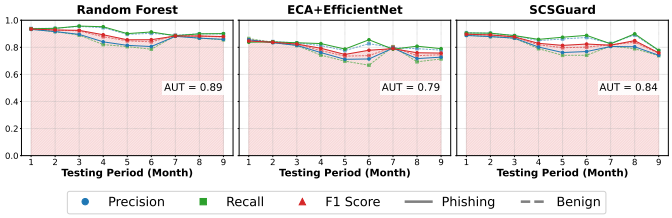

Fig. 8: Time evolution of performance metrics over nine months, with AUT for the phishing samples' F1 score.

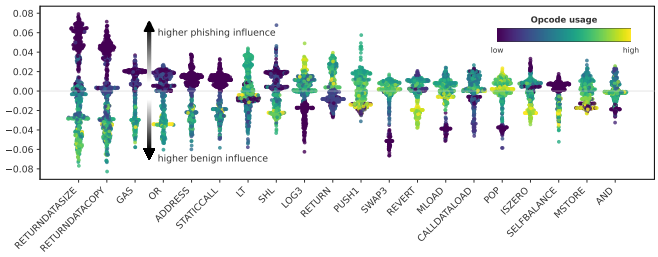

Fig. 9: HSC classifier's SHAP values of all samples of a test split (20 most influential features are shown).

V. DISCUSSION AND LESSONS LEARNED

PHISHINGHOOK is an efficient framework to systematically train, evaluate and assess ML models to detect phishing activities in the EVM. We demonstrate that it is possible to create extremely efficient phishing classification systems ($\approx 90\%$ accuracy across all the models) simply considering the contract bytecode, without collecting transaction traces by interacting with potential malicious nodes in the blockchain.

Take-away 1: Leveraging PHISHINGHOOK, we can put in place extremely effective opcode-based phishing detection systems.

To navigate the limited data availability caused by deduplication of minimal proxy contracts, we developed a post hoc analysis module to assess and generalize results from the n samples collected to the full set N of contracts deployed in the chain. Our findings showed significant differences between models of different categories, while divergences among similar models were less relevant.

Take-away 2: LMs and VMs do not perform better than HSCs on phishing detection. Differences in the same categories of models are less meaningful.

Our scalability analysis highlighted how more complex models overcome simpler HSCs in the long run at the cost of time efficiency. This conclusion meets the idea that large models likely require more data to achieve good generalization, which is a staple of machine learning research [8]. We assume that there is margin to improve the $\approx 94\%$ accuracy obtained with HSCs by enlarging the dataset.

Take-away 3: Complex models, especially LMs, scale better than HSCs. Increasing the data size, we can even improve the accuracy of the detection system using LMs.

Finally, we conducted a time-resistance analysis to evaluate PHISHINGHOOK's ability to adapt to evolving attack patterns. The results indicate that PHISHINGHOOK maintains overall robustness against temporal performance decay. Among the components, HSCs exhibited the highest stability, while LMs and VMs experienced minor fluctuations.

Take-away 4: PHISHINGHOOK is resilient to evolving threats. Despite minor fluctuations, it effectively adapts to ensure reliable long-term performance.

VI. RELATED WORK

To the best of our knowledge, PHISHINGHOOK is the first pure phishing detection framework for smart contracts based on the analysis of the opcodes. Some early classification of vulnerable and/or malicious smart contracts exist, such as [25]. We organize related work into four detection system categories: *(A)* opcode-based fraud, *(B)* transaction-based phishing, *(C)* opcode-based vulnerability and *(D)* symbolic execution and verification tools.

A. Opcode-based fraud detection systems

These systems provide efficient fraud detection systems for smart contracts basing on their bytecode analysis, without specifically address phishing activities. Alongside SCSGuard [43] and ECA+EfficientNet [85] (see §IV-B), HoneyBadger [78] employs symbolic execution and heuristics to discover honeypots in smart contracts. It takes as input the EVM bytecode and returns the detected honeypot techniques. Similarly, Al-SPSD [32] leverages ordered boosting on the features extracted from the original opcode sequences to detect Ponzi schemes in smart contracts. Finally, other approaches [26], [54] demonstrated the efficacy of supervised learning approaches, such as Random Forests (RF) [41] and k-Nearest Neighbors (k-NN) [16]. In both cases, the authors adopt feature extraction techniques on the original opcode sequences, before feeding the data to the classifier.

B. Transaction-based phishing detection systems

These tools detect phishing activities on smart contracts based on the transaction traces rather than pure bytecode analysis. Ethereum Phishing Scam Detection (Eth-PSD) [50] attempts to detect phishing scam-related transactions using several ML classifiers.TxPhishScope [40] dynamically visits suspicious websites, triggers transactions, and simulates results in order to detect phishing websites and extract related accounts automatically. Labeling transactions and labeling contracts are fundamentally different tasks. While contracts can be indirectly labeled through associated transactions, accurate analysis typically requires many transactions per contract. Additionally, replaying transactions tied to malicious actors may expose sensitive user data, creating security and privacy risks.

C. Opcode-based vulnerability detection systems

The following tools detect smart contract's code vulnerabilities (and not social engineering attacks) based on bytecode analysis, as ESCORT [72] (see §IV-B). CodeNet [46] is a CNN architecture to classify vulnerable contracts while preserving the semantics and context of the smart contract. WIDENNET [59] leverages a variant of the Wide & Deep neural network architecture [13]. In [66], the authors propose a bidirectional long-short term memory with attention mechanism (BLSTM-ATT), aimed to detect a specific vulnerability, *i.e.*, reentrancy bugs. Finally, in [53] the authors implement a 2-layer bidirectional LSTM to detect code vulnerabilities in smart contracts.

D. Symbolic execution and verification tools

Symbolic execution tools can analyze and test smart contracts deployed in the EVM by exploring their possible execution paths to detect specific vulnerabilities and bugs. DefectChecker [11] detects eight contract defects that cause unwanted behaviors of smart contracts. In [20], the authors define a smart contract defect classification scheme to evaluate the effectiveness of three popular verification tools: Mythril [21], Securify2 [76], and Slither [33]. The results show that the tools have limited detection effectiveness but are complementary in identifying different types of faults. On the other hand, [5] provides a comprehensive survey of verification methods, highlighting that most tools handle only simple contracts, with complex ones still posing challenges. In [36], the authors propose SolidiFI, a framework to evaluate six popular static analysis tools for smart contracts. It identifies numerous undetected bugs despite the tools' claimed capabilities. Finally, [39] presents Echidna, an open-source smart contract fuzzer designed to efficiently uncover real bugs with minimal user intervention and high execution speed.

VII. CONCLUSION

PHISHINGHOOK is the first comprehensive framework aimed at detecting phishing attacks on Ethereum smart contracts by analyzing opcodes and bytecode. Our extensive evaluation of 16 phishing detection models underscores the efficacy of opcode-based fraud detection, achieving high accuracy and robust performance metrics across a variety of methodologies. Our results show that more complex models (*i.e.*, LMs) do not necessarily yield better results overall but could scale better. Our key contributions include the development and public release of the largest dataset of phishing smart contracts, a detailed architectural design for the PHISHINGHOOK framework, and an in-depth experimental evaluation of multiple detection approaches. By providing an open-source, reproducible platform, we aim to foster further research and development in the domain of smart contract security. Our findings pave the way for future advancements in detecting malicious activities within decentralized environments. While the scope of PHISHINGHOOK is to detect phishing smart contracts before they are deployed, we consider live detection an interesting future work. We will explore strategies to efficiently deploy scalable phishing detection systems for smart contracts using PHISHINGHOOK. Cybersecurity firms or blockchain monitoring platforms, such as Etherscan, are potential customers of our system.

References

[1] Smart contract 0x279e2f385ce22f88650632d04260382bfb918082. etherscan.io/address/0x279e2f385cE22F88650632D04260382bFB918082. Accessed: 2025-02-25.

[2] Smart contract 0xb5e7b87e7a84276b13da3f07495e18f3e229d3a0. etherscan.io/address/0xb5e7b87e7a84276b13da3f07495e18f3e229d3a0. Accessed: 2025-02-25.

[3] M Aickin and H Gensler. Adjusting for multiple testing when reporting research results: the bonferroni vs holm methods. *Am J Public Health*, 86(5):726–728, May 1996.

[4] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. *CoRR*, abs/1907.10902, 2019.

[5] Mouhamad Almakhour, Layth Sliman, Abed Ellatif Samhat, and Abdelhamid Mellouk. Verification of smart contracts: A survey. *Pervasive and Mobile Computing*, 67:101227, 2020.

[6] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 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 6*, pages 164–186. Springer, 2017.

[7] Jiabin Bao, Debiao He, Min Luo, and Kim-Kwang Raymond Choo. A survey of blockchain applications in the energy sector. *IEEE Systems Journal*, 15(3):3370–3381, 2020.

[8] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine learning and the bias-variance trade-off. *CoRR*, abs/1812.11118, 2018.

[9] Chainabuse. Chainabuse. https://www.chainabuse.com/.

[10] Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. *ACM Computing Surveys (CSUR)*, 53(3):1–43, 2020.

[11] Jiachi Chen, Xin Xia, David Lo, John Grundy, Xiapu Luo, and Ting Chen. Defectchecker: Automated smart contract defect detection by analyzing EVM bytecode. *IEEE Trans. Software Eng.*, 48(7):2189–2207, 2022.

[12] Wubing Chen, Zhiying Xu, Shuyu Shi, Yang Zhao, and Jun Zhao. A survey of blockchain applications in different domains. In *Proceedings of the 2018 International Conference on Blockchain Technology and Application*, pages 17–21, 2018.

[13] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In Alexandros Karatzoglou, Balázs Hidasi, Domonkos Tikk, Oren Sar Shalom, Haggai Roitman, Bracha Shapira, and Lior Rokach, editors, *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, DLRS@RecSys 2016, Boston, MA, USA, September 15, 2016*, pages 7–10. ACM, 2016.

[14] Norman Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin*, 114(3):494–505, 1993.

[15] William Jay Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, third edition, 1999.

[16] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[17] Cyvers. Malcon api: Advanced malicious contract detection & prevention. https://cyvers.ai/malconapi.

[18] Natarajan Deepa, Quoc-Viet Pham, Dinh C Nguyen, Sweta Bhattacharya, B Prabadevi, Thippa Reddy Gadekallu, Praveen Kumar Reddy Maddikunta, Fang Fang, and Pubudu N Pathirana. A survey on blockchain for big data: Approaches, opportunities, and future directions. *Future Generation Computer Systems*, 131:209–226, 2022.

[19] Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, 2006.

[20] Bruno Dia, Naghmeh Ivaki, and Nuno Laranjeiro. An empirical evaluation of the effectiveness of smart contract verification tools. In *2021 IEEE 26th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 17–26. IEEE, 2021.

[21] ConsenSys Diligence. Mythril: Security analysis tool for ethereum smart contracts. https://github.com/ConsenSys/mythril. Accessed: 2024-11-25.

[22] Alexis Dinno. Nonparametric pairwise multiple comparisons in independent groups using dunn's test. *The Stata Journal*, 15(1):292–300, 2015.

[23] 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. 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, 2021.

[24] Olive Jean Dunn. Multiple comparisons using rank sums. *Technometrics*, 6(3):241–252, 1964.

[25] Thomas Durieux, João F Ferreira, Rui Abreu, and Pedro Cruz. Empirical review of automated analysis tools on 47,587 ethereum smart contracts. In *Proceedings of the ACM/IEEE 42nd International conference on software engineering*, pages 530–541, 2020.

[26] Tahmina Ehsan, Muhammad Usman Sana, Muhammad Usman Ali, Elizabeth Caro Montero, Eduardo Silva Alvarado, Sirojiddin Djuraev, and Imran Ashraf. Securing smart contracts in fog computing: Machine learning-based attack detection for registration and resource access granting. *IEEE Access*, 12:42802–42815, 2024.

[27] Ethereum. evmdasm library. https://github.com/ethereum/evmdasm. Accessed: 2024-11-25.

[28] Etherscan. Ether total supply and market capitalization chart. https://etherscan.io/stat/supply. Accessed: 2025-03-28.

[29] Etherscan. Etherscan.io website. https://etherscan.io/. Accessed: 2024-11-25.

[30] EVM.codes. Evm opcodes (shanghai update). https://www.evm.codes/?fork=shanghai. Accessed: 25-Nov-2024.

[31] Hugging Face. Vit-base-patch16-224, 2024. Accessed: 2024-12-01.

[32] Shuhui Fan, Shaojing Fu, Haoran Xu, and Xiaochun Cheng. Al-spsd: Anti-leakage smart ponzi schemes detection in blockchain. *Information Processing & Management*, 58(4):102587, 2021.

[33] Josselin Feist, Gustavo Grieco, and Alex Groce. Slither: A static analysis framework for smart contracts. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pages 8–15, 2019.

[34] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. Codebert: A pre-trained model for programming and natural languages. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 1536–1547. Association for Computational Linguistics, 2020.

[35] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

[36] Asem Ghaleb and Karthik Pattabiraman. How effective are smart contract analysis tools? evaluating smart contract static analysis tools using bug injection. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 415–427, 2020.

[37] Gibran Gomez, Kevin van Liebergen, Davide Sanvito, Giuseppe Siracusano, Roberto Gonzalez, and Juan Caballero. Sorting out the bad seeds: Automatic classification of cryptocurrency abuse reports. *arXiv preprint arXiv:2410.21041*, 2024.

[38] Google. Google bigquery documentation. https://cloud.google.com/bigquery/docs. Accessed: 2024-11-25.

[39] Gustavo Grieco, Will Song, Artur Cygan, Josselin Feist, and Alex Groce. Echidna: effective, usable, and fast fuzzing for smart contracts. In *Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis*, pages 557–560, 2020.

[40] Bowen He, Yuan Chen, Zhuo Chen, Xiaohui Hu, Yufeng Hu, Lei Wu, Rui Chang, Haoyu Wang, and Yajin Zhou. 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, page 120–134, New York, NY, USA, 2023. Association for Computing Machinery.

[41] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, 1995.

[42] Jason Hong. The state of phishing attacks. *Communications of the ACM*, 55(1):74–81, 2012.

[43] Huiwen Hu, Qianlan Bai, and Yuedong Xu. 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, pages 1–6. IEEE, 2022.

[44] TonTon Hsien-De Huang and Hung-Yu Kao. R2-D2: color-inspired convolutional neural network (cnn)-based android malware detections. In 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 International Conference on Big Data (IEEE BigData 2018), Seattle, WA, USA, December 10-13, 2018*, pages 2633–2642. IEEE, 2018.

[45] HuggingFace. Huggingface transformers library. https://huggingface.co/docs/transformers/index.

[46] Seon-Jin Hwang, Seok-Hwan Choi, Jinmyeong Shin, and Yoon-Ho Choi. Codenet: Code-targeted convolutional neural network architecture for smart contract vulnerability detection. *IEEE Access*, 10:32595–32607, 2022.

[47] TRM Investigations. Grim finance hacked: 600 million in crypto stolen in december. https://www.trmlabs.com/post/grim-finance-hacked-600-million-in-crypto-stolen-in-december.

[48] Athirai A. Irissappane, Hanfei Yu, Yankun Shen, Anubha Agrawal, and Gray Stanton. Leveraging GPT-2 for classifying spam reviews with limited labeled data via adversarial training. *CoRR*, abs/2012.13400, 2020.

[49] Tengyun Jiao, Zhiyu Xu, Minfeng Qi, Sheng Wen, Yang Xiang, and Gary Nan. A survey of ethereum smart contract security: Attacks and detection. *Distrib. Ledger Technol.*, 3(3), September 2024.

[50] Arkan Hammoodi Hasan Kabla, Mohammed Anbar, Selvakumar Manickam, and Shankar Karupayah. Eth-psd: A machine learning-based phishing scam detection approach in ethereum. *IEEE Access*, 10:118043–118057, 2022.

[51] W. H. Kruskal and W. A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952.

[52] Scikit learn developers. Permutation feature importance, 2024. Accessed: 2025-02-24.

[53] Peiqiang Li, Guojun Wang, Xiaofei Xing, Jinyao Zhu, Wanyi Gu, and Guangxin Zhai. A smart contract vulnerability detection method based on deep learning with opcode sequences, Sep 2024.

[54] Derek Liu, Francesco Piccoli, and Victor Fang. Machine learning approach to identify malicious smart contract opcodes: A preliminary study. *JPS Conference Proceedings (Blockchain Kaigi 2023)*, 43:011002, 2023.

[55] Jiming Liu and Yiming Ye. *Introduction to e-commerce agents: Marketplace marketplace solutions, security issues, and supply and demand*. Springer, 2001.

[56] 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. Codexglue: A machine learning benchmark dataset for code understanding and generation. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021.

[57] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4765–4774, 2017.

[58] Rodrigo Frassetto Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. Document ranking with a pretrained sequence-to-sequence model. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 708–718. Association for Computational Linguistics, 2020.

[59] Samuel Banning Osei, Zhongchen Ma, and Rubing Huang. Smart contract vulnerability detection using wide and deep neural network. *Science of Computer Programming*, 238:103172, 2024.

[60] Florian Pargent, Florian Pfisterer, Janek Thomas, and Bernd Bischl. Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features. *Comput. Stat.*, 37(5):2671–2692, 2022.

[61] Daejun Park, Yi Zhang, Manasvi Saxena, Philip Daian, and Grigore Roşu. A formal verification tool for ethereum vm bytecode. In *Proceedings of the 2018 26th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, pages 912–915, 2018.

[62] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. TESSERACT: eliminating experimental bias in malware classification across space and time. In Nadia Heninger and Patrick Traynor, editors, *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 729–746. USENIX Association, 2019.

[63] Daniel Perez and Benjamin Livshits. Smart contract vulnerabilities: Vulnerable does not imply exploited. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1325–1341, 2021.

[64] Ethereum Improvement Proposals. Erc-1167: Minimal proxy contract. https://eips.ethereum.org/EIPS/eip-1167/. Accessed: 2024-11-25.

[65] Peng Qian, Jianting He, Lingling Lu, Siwei Wu, Zhipeng Lu, Lei Wu, Yajin Zhou, and Qinming He. Demystifying random number in ethereum smart contract: taxonomy, vulnerability identification, and attack detection. *IEEE Transactions on Software Engineering*, 49(7):3793–3810, 2023.

[66] Peng Qian, Zhenguang Liu, Qinming He, Roger Zimmermann, and Xun Wang. Towards automated reentrancy detection for smart contracts based on sequential models. *IEEE Access*, 8:19685–19695, 2020.

[67] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[68] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020.

[69] Phylum Research. Typosquat campaign targeting npm developers. https://blog.phylum.io/supply-chain-security-typosquat-campaign-targeting-puppeteer-users/.

[70] Pasquale De Rosa, Simon Queyrut, Yerom-David Bromberg, Pascal Felber, and Valerio Schiavoni. Phishinghook: Catching phishing ethereum smart contracts leveraging evm opcodes. https://doi.org/10.5281/zenodo.14260284, Mar 2025.

[71] The scikit-learn developers. *Supervised Learning - scikit-learn 1.5.0 documentation*, 2024. Accessed: 2024-12-01.

[72] Christoph Sendner, Huili Chen, Hossein Fereidooni, Lukas Petzi, Jan König, Jasper Stang, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. Smarter contracts: Detecting vulnerabilities in smart contracts with deep transfer learning. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2023.

[73] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3-4):591–611, 12 1965.

[74] David Siegel. Understanding the dao attack. https://www.coindesk.com/learn/understanding-the-dao-attack.

[75] Solidity Team. Solidity language documentation. https://soliditylang.org/. Accessed: 25-Nov-2024.

[76] ETH Zurich SRI Lab. Securify 2.0. https://github.com/eth-sri/securify2. Accessed: 2025-03-28.

[77] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019.

[78] Christof Ferreira Torres, Mathis Steichen, and Radu State. The art of the scam: Demystifying honeypots in ethereum smart contracts. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1591–1607, Santa Clara, CA, August 2019. USENIX Association.

[79] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123–147, 2019.

[80] Maddipati Varun, Balaji Palanisamy, and Shamik Sural. Mitigating frontrunning attacks in ethereum. In *Proceedings of the Fourth ACM International Symposium on Blockchain and Secure Critical Infrastructure*, pages 115–124, 2022.

[81] Vyper Project. Viper language documentation. https://docs.vyperlang.org/en/stable/. Accessed: 25-Nov-2024.

[82] Qilong Wang, Banggu Wu, Pengfei Zhu, Peihua Li, Wangmeng Zuo, and Qinghua Hu. 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*, pages 11531–11539. Computer Vision Foundation / IEEE, 2020.

[83] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[84] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger (shangai version). *Ethereum project yellow paper*, pages 1–42, 2024.

[85] Xuanchen Zhou, Wenzhong Yang, Liejun Wang, Fuyuan Wei, KeZiErBieKe HaiLaTi, and Yuanyuan Liao. The detection of fraudulent smart contracts based on eca-efficientnet and data enhancement. *Computers, Materials and Continua*, 77(3):4073–4087, 2023.