Features Tell a Tale: Multi-Method Feature Analysis in Smart Contract Security

Sarang Pratap Chamola

Fulda University of Applied Sciences, Fulda, Germany
sarang-pratap.chamola@informatik.hs-fulda.de

Abstract. Smart contracts are conditional codes written for critical transactions, but frequently have undetected flaws or exceptions that can increase vulnerability. In our work, we found that contracts with complex architecture, many external functions, and complicated internal structure are 27% more likely to have vulnerabilities than simple contracts. Here, we have implemented three different feature importance algorithms (Gini, XGBoost, and SHAP) and analyzed 111,897 Solidity contracts, to find the degree to which different features, or variables, contribute to the predictions of a machine learning model. The aim of this research is to find the features which play an important role in recognizing the vulnerability in smart contracts.

1 Introduction

Smart contracts have eased the use case of blockchain technology by allowing agreements to be executed automatically without intermediaries. Smart contracts have many use cases in real life. The following are some use cases:

Health Care Industry: Digital Healthcare platforms are used now days for patients privacy and record management and care automation, one such platform is "Safe" [1].

Music Industry: Smart contract can be used to make royalty payments easier for artist.

Insurance Sector: The insurance world has a lot of third-party associated before settling out an insurance, automation of services using smart contract can reduce the premium price of insurances and allow users to get faster settlements of claims.

Supply Chain Management: The supply chain domain has issues with transparency and tracking. Smart contracts can come in handy without the need for third parties to increase transparency by using predetermined clauses in contract to ensure smooth operations.

Storing Digital Identity: When smart contracts are used in different online services, it is really important to learn about a person's reliabilities without knowing

their whole identity. Smart contract can be written to create some kind of credit score that can allow any specific seller to trust and reduce any risk of fraud.

Almost 13.56 million [2] smart contracts have been deployed from the beginning of January 2025 to October 2025. Figure 1 [2] represents the total number of contracts deployed on the Ethereum network for the above mentioned time frame. Which shows that the use case of smart contracts is on a rise and many new smart contracts are being deployed every day.

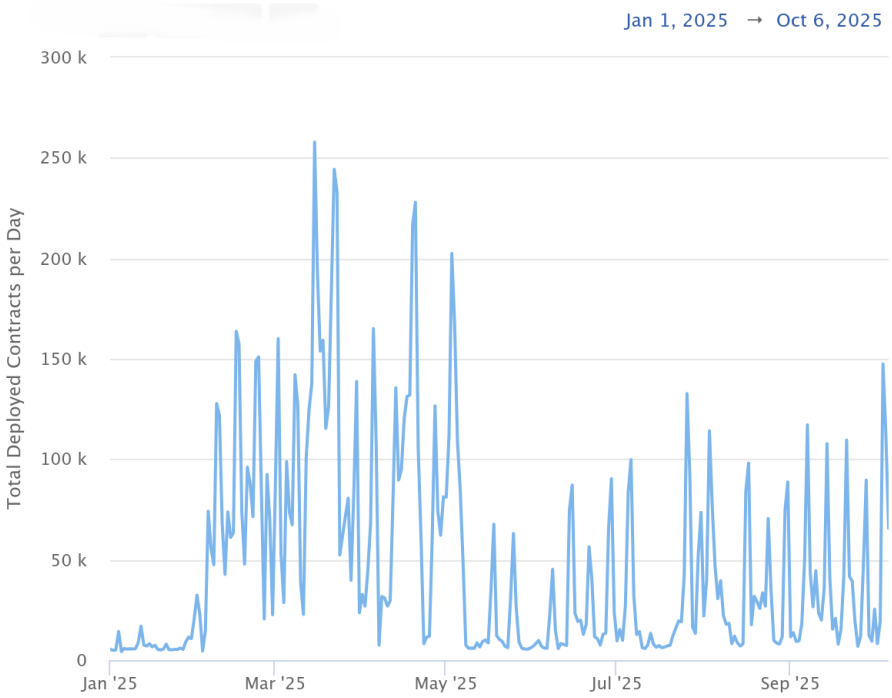

Fig. 1. Ethereum Daily Deployed Contracts Chart

However, once a contract is deployed, any vulnerabilities cannot be easily fixed due to the immutable nature of a blockchain, which makes security crucial. A hundreds of million dollars were lost in the incidents from the past such as Dao attack in 2016 [3] and Parity wallet freeze in 2017 [4] . Due to these incidents the urgent need of finding the cause root of vulnerabilities became important and also to further prevent them.

There are two types of methods for smart contract security analysis, one is a statistical analysis tool and another is machine learning-based detection systems. Statistical tool such as Mythril [5] and Slither [6] usually follow a set of predefined patterns to find bugs for different types of vulnerabilities in the code written for

the smart contracts. Since they work with predefined set of patterns they struggle with new types of vulnerabilities. On the other hand machine learning methods take use of algorithmic approach to check for the vulnerability analysis but not always help in understanding what causes the vulnerability in the first place.

The idea is to understand the characteristics of features that actually help machine learning algorithms to predict vulnerabilities. Most studies use a few sets of machine learning algorithms with few features which do not provide a total view of vulnerabilities. There is no systematic review done on different methods of feature importance taking into account newly developed methods to identify the features associated with vulnerability risk, which is important for new tools and also for manual analysis.

This research tackles these issues by presenting the large-scale, multi-method feature importance analysis for detecting smart contract vulnerabilities.

1.1 The aim and objectives of the study

This work aims to create a multiple method feature importance analysis to get the understanding of features that helps the most in vulnerability detections and have the highest level of importance in the machine learning model. We have taken BCCC-SCsVul-2024 dataset from a research article on smart contract vulnerability [7], which contains 111897 Solidity source code samples.The data is then pre-processed using python and further analysis is drawn from it. Here we introduce the use of SHAP (SHapley Additive exPlanation) values [8] as a measure of feature importance. This work focuses on reducing the vulnerabilities by identifying the root cause of attacks on smart contract and try to answer due to which features different machine learning model makes a certain prediction.

2 Background and Related work

This chapter focus on the current research related to smart contracts and offers deep dive at the theoretical basis and current research related to multi-method feature importance analysis in the security of smart contracts. We investigate the development of methods for detecting vulnerabilities, deep dive into the methodological principles behind feature importance analysis, and understand the current status of research on the application of machine learning methods in blockchain security.

Understanding Blockchain Technology Blockchain technology serves as the foundational infrastructure for smart contracts. At its core, a blockchain is a distributed, immutable digital ledger that maintains a continuously growing list of records, called blocks, which are linked and secured using cryptographic principles [9].

Understanding Ethereum Technology Vitalik Buterin, the known creator of Ethereum launched it in 2015, getting attention from the all across the work and currently one of the famous blockchain after Bitcoin. While Bitcoin primarily functions as a digital currency, Ethereum was designed as a programmable blockchain platform capable of executing complex applications written in a specific programming language to automate the transactions securely [10].

2.1 Smart Contracts: Self executed written agreements

Smart contracts are high-level programs that execute a set of functions automatically, without the need for for any third party or person to verify. These agreements are recorded on the blockchain and operate exactly as coded, offering transparency, efficiency, and cost savings.

Key Characteristics:

- **Automatic Execution**: Smart contracts automatically trigger when specified conditions are fulfilled, removing the requirement for manual handling or third-party enforcement [11].
- **Immutability**: After they are deployed, smart contracts cannot be altered, which assures all participants that the contract's behavior will remain consistent.
- **Consistent Outcomes**: Smart contracts yield the same results every time they are given identical inputs, guaranteeing predictable results across all network participants.
- **Clarity**: The code of the contract is openly accessible on the blockchain, enabling anyone to verify its functionality and terms.
- **Coding Language**: The majority of contracts are developed using Solidity[12],a statically-typed programming language specifically designed for blockchain developments. Solidity also supports object-oriented concepts and also provides built-in functions for blockchain operations [13].

2.2 Importance of Features in Machine Learning

Feature importance is a fundamental technique in machine learning that provides insights on how much each input feature contributes to a model prediction,again provides insights on how to treat the variable better for model development and optimization.Feature importance is a fundamental tool for machine learning applied in security use-cases, since it not only enhance security, but also develops an understanding of what actually is driving vulnerability in the first place. Many algorithms are used to determine feature importance, among them there are few widely used algorithms such as Tree-based impurity measures using gini, gradient-boosting gain metrics using XGBoost, and game-theoretic contribution analysis using SHAP analysis. The following subsection describes the mathematical foundation of each algorithm:

Method 1: Gini Importance via Random Forest

Mathematical Foundation: *Gini Impurity is a measurement used to build Decision Trees to determine how the features of a dataset should split nodes to form the tree. For a node t with class distribution p(i/t), the Gini impurity is: [14][15]*

$$\text{Gini}(t) = 1 - \sum_i p^2(i \mid t)$$

The Gini importance of feature j is computed as:

$$GI(j) = \sum_t \frac{n_t}{N} \left[\text{Gini}(t) - \sum_{c \in \text{children}(t)} \frac{n^c}{n_t} \text{Gini}(c) \right]$$

where:

- n_t = number of samples reaching node t
- N = total number of samples
- The summation occurs over all nodes t where feature j is used for splitting

Method 2: XGBoost Gain-based Importance

Mathematical Foundation: *XGBoost employs gradient boosting with regularization. Gain implies the improvement in accuracy (loss reduction) added in the prediction by using a feature to the branches it is used in. It measures how much each feature adds to the model's predictive power, with higher values indicating more important features [16].:*

$$\text{Gain}(j) = \sum_{t \in T(j)} \frac{n_t}{N} \left[L(t) - \sum_{c \in \text{children}(t)} \frac{n^c}{n_t} L(c) \right]$$

where:

- T(j) = set of all tree nodes using feature j for splitting
- L(t) = loss function value at node t
- The gain represents the reduction in loss when splitting on feature j

The objective function optimized by XGBoost is:

$$\text{Obj}^{(t)} = \sum_i l\big(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\big) + \Omega(f_t) + \text{constant}$$

where l is the loss function and Ω is the regularization term.

Method 3: SHAP (SHapley Additive exPlanations)

Mathematical Foundation: *SHAP values are based on cooperative game theory, specifically the Shapley values. in the team game where each player performs differently, the shap values are provided on the basis of how much credit or blame a player deserves, similarly here it provides the importance to features on the basis of their performance in correct predictions. For a feature i, the SHAP value ϕ_i represents its marginal contribution [8]:*

$$\varphi_i \;=\; \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!\,\big(|F| - |S| - 1\big)!}{|F|!}\,\Big[f(S \cup \{i\}) - f(S)\Big]$$

where:

- F = set of all features
- S = subset of features not including i
- f(S) = model output when using feature subset S
- The summation is over all possible subsets S

Core Value of Feature Importance Feature importance provides a very detailed insight into model behavior by calculating scores of the features that represent the contribution of each feature. These scores answer the question: What characteristics drive the predictability and decisions of my model? The technique works by measuring the degree of usefulness of every variable in the data set for current models and prediction. For example, when predicting a car resale value, its performance, quality, and year of manufacture are likely to receive a high importance score, while few features such as the auto dealership, the owner's favorite color, would score near zero values.

2.3 Previous work and Insights.

Feature importance provides hidden relationships between feature and target variables. This is valuable when the model is also a means of learning about data patterns and not just to make predictions. This analysis is crucial for building new security measures taking into account features with high-importance scores.

Some previous study on Ethereum analysis smart contract vulnerability detection has been conducted in this paper "An interpretable model for large-scale smart contract vulnerability detection "[17]. The paper presents feature importance using shap analysis in vulnerability analysis section. This work does not use a dataset with broad range of vulnerabilities for an example BCCC-SCsVul-2024 dataset to get more effective features and also does rely on a single feature importance algorithm, which here this research tries to fill the gap by using more data to identify more effective features which increase the risk of vulnerabilities with less bias using multi-model feature importance.

The work of [7] introduces the BCCC-SCsVul-2024 dataset and focuses on identifying 11 different vulnerabilities in Smart contracts, which has a sample size of 111,897 instances this dataset, that covers a broad range of 11 different potential vulnerabilities from several sources, including gas exceptions, mishandled exceptions, denial of service, timestamps, external bugs, reentrancy and Non-Vulnerable which are briefly discussed in section 3.1. The work [7] trained new machine learning tools and also used their dataset on previous two benchmark tools, Mythril [5] and Slither [6], but they did not perform feature importance using different methods to identify the important features.

The work uses BCCC-SCsVul-2024 dataset to develop a multi-method feature importance analysis to find these features which likely cause different vulnerabilities.

3 Methodology

Our work utilizes a multi-model framework approach combining three different feature importance methods for further analyzing the responsible features for machine learning algorithms to classify for vulnerabilities in smart contracts. The dataset, BCCC-SCsVul-2024 [7] contains 111,897 samples of smart contracts across 12 different vulnerability categories, and 240 features were engineered for analysis. A three methods framework ensembling includes Random Forest Gini Importance (500 estimators with balanced class weighting), XGBoost Gain Analysis (500 boosting iterations with a learning rate of 0.05), and SHAP Values (TreeExplainer with 1,000 strategic samples). After the individual scores, all scores were normalized and a final score was derived.

3.1 Data Processing and Feature Engineering

Dataset Preparation The BCCC-SCsVul-2024 dataset containing 111,897 smart contract samples with 253 initial features which were preprocessed by dropping two id columns and encoding all 12 class columns into a single target variable. The dataset comprises multiple vulnerability classes as shown in Figure 2, and have been explained below:

Target vulnerability classes

- **Class01: ExternalBug**
  Vulnerabilities known as external bug arise when smart contracts connect with other contract or fail to properly handle external function. This includes situation in which external function call fails without proper error handling, leading to inconsistent contract states. The study shows that this vulnerability had 3,604 samples in their dataset.
- **Class02: GasException**
  As the electricity is measured in KWh, similarly computation and storage

used in Ethereum is measured in Gas [18]. So GasException vulnerabilities appear when smart contracts consume more gas then their limits during execution, which sometimes causing transactions to fail due to exceeding gas limits. Dataset contains 6,879 samples of this exception, this vulnerability can be targeted by attackers to create denial-of-service conditions. Gas-related vulnerabilities occur when contracts contain unbounded loops, expensive operations, or patterns that can be used to force contracts to run out of gas.

- **Class03: MishandledException**
  This type of vulnerability appears when a smart contract is not able to handle exceptions or edge case conditions during execution. This also includes cases where exceptions are caught but are not handled properly leading to a faulty contract behavior. The dataset contains 5,154 samples of this vulnerability type. Attackers can exploit such cases such as DOS (denial of service), on the on-going contract like if a malicious bidder in an auction becomes a leader, he can remain the leader forever because he can prevent anyone else from successfully calling the function bid via a costly fallback function [19].

- **Class04: Timestamp**
  Timestamp Dependence vulnerabilities happen when smart contracts are dependent on some time. Attackers can change the timestamps of the contract logic which depends on precise time. The applications such as lotteries and auctions can be potentially attacked by this vulnerability. The dataset contains 2,674 samples of this vulnerability type.

- **Class05: TransactionOrderDependence**
  This type of vulnerabilities can appear when the result of a smart contract depends on the order of transaction. The dataset contains 3,562 samples of this vulnerability. An unfair advantage can be taken by submitting a different transaction with the higher gas fees to run first.

- **Class06: UnusedReturn**
  This type of vulnerability occurs when a function in smart contract is not able to send proper return value to another contract. This dataset contains 3,229 samples of unused return type.

- **Class07: WeakAccessMod**
  If a function or a variable is not in the correct access control, this type of vulnerability can happen allowing attackers to executee private functions which are strictly for owners. The dataset contains 1,918 samples of this vulnerability.

- **Class08: CallToUnknown**
  Call to Unknown vulnerabilities happen when smart contracts make calls to unverified or potentially malicious external addresses. With 11,131 this vulnerability can lead to reentrancy attacks or unexpected behavior when

calling untrusted contracts. It's related to making external calls without proper validation of the target address.

- **Class09: DenialOfService**
  Denial of Service (DoS) vulnerabilities allow attackers to make smart contracts unusable by consuming critical resources like gas, CPU cycles, or storage. The dataset contains 12,394 samples, making it one of the most common vulnerabilities. DoS attacks can be achieved through gas limit manipulation, unbounded loops, or forcing contracts into states where legitimate users cannot access their functions.
- **Class10: IntegerUO (Integer Underflow/Overflow)**
  Integer Underflow/Overflow vulnerabilities occur when arithmetic operations exceed the maximum or minimum values that can be stored in integer variables. With 16,740 samples, this is a significant vulnerability class that can allow attackers to manipulate token balances or other numeric values. While Solidity 0.8.0+ includes automatic overflow protection, older contracts remain vulnerable.
- **Class11: Reentrancy**
  Reentrancy vulnerabilities happen when external contracts can call back into the original contract before the first call completes, potentially manipulating the contract's state. With 17,698 reentrancy is one of the most critical and well-known smart contract vulnerabilities. Famous attacks like "The DAO" [3] exploit this vulnerability by repeatedly calling withdrawal functions before balance updates.
- **Class12: Non Vulnerable (Secure)**
  Non Vulnerable represents secure smart contracts that don't contain any of the identified vulnerability patterns. The dataset includes 26,914 secure samples, which serve as the baseline for comparison against vulnerable contracts. These contracts are the baseline which follow security best practices and proper secure programming techniques.

Feature Types The dataset includes six different feature categories, AST, ABI Specification, bytecode, op-code,source code and solidity information of Smart contracts. Table 1 provides a summary of the features.

- **AST(Abstract Syntax Tree)**: Abstract Syntax is a data structure in computer science that captures the syntactic structure of the source code in a formal language [20]. It contains information about the program, such as the position of each element in the source code when analyzed by the compiler [21].
- **ABI specifications**: The Ethereum ecosystem relies on the contract ABI, which is a common interface between two different program modules, often between the operating system and user program. Ethereum treats Smart contracts as a byte of program. As a user can only work with high-level lan-

guage, so ABI documentation ensures that the data is encoded and decoded correctly while doing translations between user intended function calls [22].

- **Bytecode:** Bytecode is a form of computer object code that an interpreter converts into binary machine code, which is further fetched by the computer hardware processor to read [23]. The solidity code is translated into bytecode and contains binary information for the computer [24].
- **Opcode:** Opcode holds a unique position in machine learning instruction sequence, it holds the one-byte sequence of code that specifies a set of operations, for example, addition, multiplication, etc [25]. [26] repository contains all the opcodes used in Ethereum.
- **Source code:** Source code is the human written instructions in a programming language that perform a set of tasks. Each type of feature within this category offers unique and essential insights that can help in the vulnerability detection process [7]. It contains features such as number of functions, total lines of code, total comment lines, empty lines in database that is all the information about the written code.
- **Solidity Information**: Solidity information is a non-compiler based features, which contains all the information about the count of functions that were used in smart contract code [7].

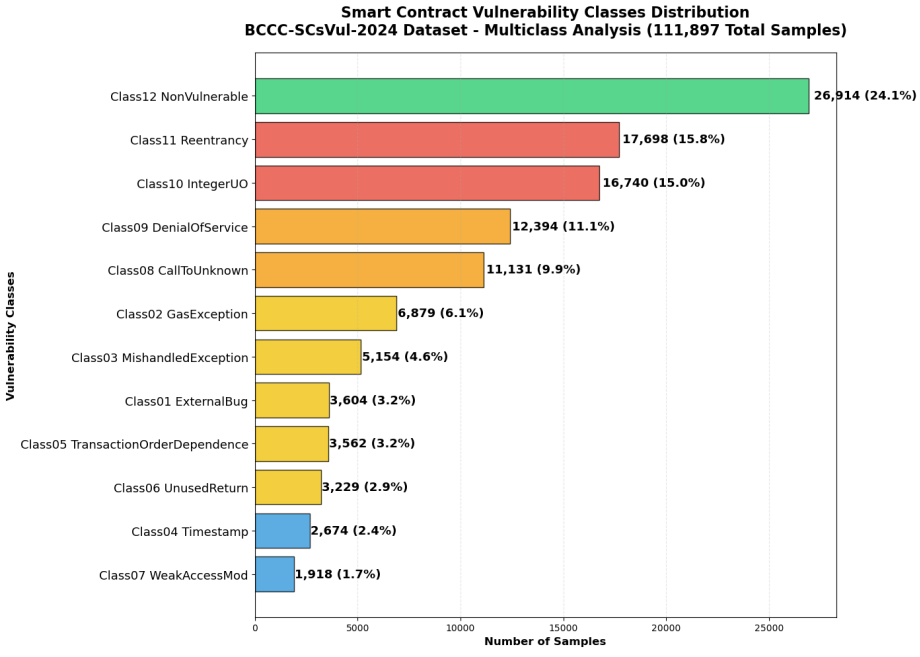

Fig. 2. Class distribution of smart contract vulnerabilities

Table 1. Feature summary

Features Type	**Description**
AST	The Abstract Syntax Tree (AST) features represent the structural elements of source code.
ABI specifications	Provides a standard way for external interaction and contract-to-contract communication in the Ethereum ecosystem by encoding data according to its type.
Bytecode	A form of computer object code that an interpreter converts into binary machine code.
Opcode	Represents a specific operation in the environment, e.g., addition or multiplication, with 256 possible opcodes.
Source code / Code Metrics	Converts raw text into numerical representations so that machine learning algorithms can process it more easily.
Solidity Information / Functional Features	Includes the count of functions used in smart contracts along with corresponding instructions.

3.2 Data pre-processing

To ensure compatibility across different machine learning frameworks, particularly XGBoost which has strict feature naming requirements, a function $F'(i)$ was implemented that ensures all the variables with invalid characters in their names are replaced by underscores.

$$F'(i) = \text{sanitize}(F(i)) \tag{1}$$

where $F'(i)$ replaces invalid characters `[`, `]`, `<`, `>` with underscores.

The 12 different vulnerability classes were encoded to a single categorical label using:

$$y = \arg\max(C_1, C_2, \ldots, C_n) \tag{2}$$

where C_i represents the class probability for vulnerability type i and the class with highest probability is stored in column y.

Abstract Syntax Tree (AST) node type features, originally in string format, were encoded using Label Encoding to preserve ordinality.

Id and AST source ID columns were dropped as they contain the unique identification for the smart contracts and abstract syntax tree of that smart contract.

3.3 Multi-Method Importance Analysis

Feature importance analysis in complex datasets, such as those used for smart contract vulnerability detection, presents significant methodological challenges. Algorithms often prioritize features according to their internal mechanisms instead of actual predictive relevance [27] [28]. Consequently, individual methods

may fail to identify critical features or may likely highlight features that conform to their specific mathematical frameworks. The paper [29] published in Proceedings of the National Academy of Sciences, presents a series of "impossibility theorems" for feature importance methods. It shows that no single feature importance method is self sufficient and can generate spurious results.

To solve this problem, a framework has been created by using three different algorithms to measure feature importance, for which the foundation has been discussed in section 2.2. Thereafter, the research looks for agreement across different mathematical approaches. If these varied algorithms consistently highlight the same features, we can be more confident that those features truly help in prediction.

Each method highlights a different way that features affect model performance. When these methods agree, it gives us stronger evidence about which features matter most than relying on just one method.

Importance Score Normalization and Aggregation

Each importance score was normalized to generate meaningful comparisons across methods with different abilities.

$$I'_{j,m} = \frac{I_{j,m}}{\max\left(I_{1,m}, I_{2,m}, \ldots, I_{p,m}\right)}$$

where $I_{j,m}$ represents the raw importance of the feature j from method m given p overall features .

The final aggregated importance score was computed as:

$$\text{Importance}_{\text{avg}}(j) = \frac{1}{3}\left[I'_{j,\text{Gini}} + I'_{j,\text{XGB}} + I'_{j,\text{SHAP}}\right]$$

Methodological Justification This tri-method approach was used for the following known reasons:

1. **Conceptual:** Each importance method highlights a different perspective of feature importance- Gini looks at the split quality, XGBoost measures the predictive gain, and SHAP on how much a feature contributes to the decision of prediction.
2. **Bias :** Single-method results can be a form of algorithmic biases [29]. The ensemble approach decreases method-specific biases and provides more normalized rankings.
3. **Theoretical Diversity:** This approach brings together Gini (information theory), XGBoost (optimization), and SHAP (game theory) to give a thorough way of evaluating features.

4 Results and Discussion

This section will explore the results of the proposed model.Starting with the top performing features, the research also deep dives in method-specific insights following with an analysis by feature category and feature ranking stability across all the models.

4.1 Feature Importance Analysis Results

Top-Performing Features : Presented multi-method analysis identified a general set of important features for smart contract vulnerabilities. Table 2 includes the top 15 features ranked by average normalized importance across all methods.

Table 2. Top 15 features ranked by average normalized importance across all methods.

Feature	Gini Importance	XGB Gain	SHAP Importance	Gini Norm.	XGB Norm.	SHAP Norm.	Avg. Importance
AST_Features_ast_len_exportedSymbols	0.0128	148.1409	0.3197	0.2455	1.0000	1.0000	0.7485
AST_Features_ast_len_nodes	0.0141	72.1510	0.2622	0.2707	0.4870	0.8201	0.5259
Opcode_Count_Features_STOP	0.0048	112.1958	0.1631	0.0923	0.7574	0.5100	0.4532
Lines_of_Code_0	0.0519	9.0263	0.0330	1.0000	0.0609	0.1033	0.3881
Lines_of_Code_1	0.0508	6.4708	0.0399	0.9784	0.0437	0.1247	0.3823
Lines_of_Code_3	0.0461	6.4872	0.0456	0.8867	0.0438	0.1427	0.3577
Functional_Features_0	0.0445	7.7978	0.0494	0.8561	0.0526	0.1544	0.3544
Solidity_Features_4	0.0316	19.4304	0.0742	0.6087	0.1312	0.2320	0.3240
Duplicate_Lines_Count	0.0388	6.7167	0.0361	0.7472	0.0453	0.1130	0.3018
Functional_Features_2	0.0326	6.9452	0.0518	0.6286	0.0469	0.1620	0.2791
AST_Features_ast_id	0.0217	14.2450	0.0951	0.4176	0.0962	0.2976	0.2705
Functional_Features_1	0.0318	5.4999	0.0502	0.6116	0.0371	0.1571	0.2686
Lines_of_Code_2	0.0335	5.3972	0.0375	0.6449	0.0364	0.1173	0.2662
AST_nodetype_encoded	0.0055	100.9687	0.0012	0.1066	0.6816	0.0037	0.2640
Contract_Information_0	0.0326	11.1579	0.0281	0.6271	0.0753	0.0879	0.2635

The analysis highlights that `AST Features_ast_len_exportedSymbols` came out as the most influential feature with an importance of 0.74, maintaining the maximum normalized scores in both XGBoost Gain (1.0) and SHAP importance (1.0). This feature represents the count of the exported symbols in the abstract Syntax Tree, and is the most stable across the different methodologies.Thus,it follows that this feature has a potential impact in classification of vulnerabilities.

`AST Features_ast_len_nodes` comes out to be the second-best feature (0.52),and has shown a significant high performance in SHAP(0.82) and a decent importance in the Gini-based ranking (0.27). This feature represents the count of immediate descendant nodes within the AST structure.

`Opcode Count Features_STOP` appears to be the third-best feature, indicating its crucial role in classification of vulnerability and importance in writing smart contracts. This feature represents the number of STOP instructions in bytecode.

In the same table 2, it is important to note that many of `Lines of Code` Features (`Lines_of_Code_0, Lines_of_Code_1, Lines_of_Code_3`) have higher importance levels, which suggests that lines of code play a vital role in securing smart contracts, i.e. larger and more complex or some bug written solidity codes can be more prone to attacks. The feature `Lines_of_code_0` represents the comment lines, that is, lines with comments / explanations, feature `Lines_of_code_1` represents the blank lines, that is empty lines with no code or comments and `Lines_of_code_2` represents the number of lines that are repeated in the code.

4.2 Method-Specific Insights

Gini Importance Patterns: The Random Forest computed by gini importance shown in Figure 3, shows a high preference for structural code metrics, with `Lines of Code_0` having the highest score (0.0519). This aligns with the tendency of decision tree algorithms to favor features that provide clear split criteria for class separation.

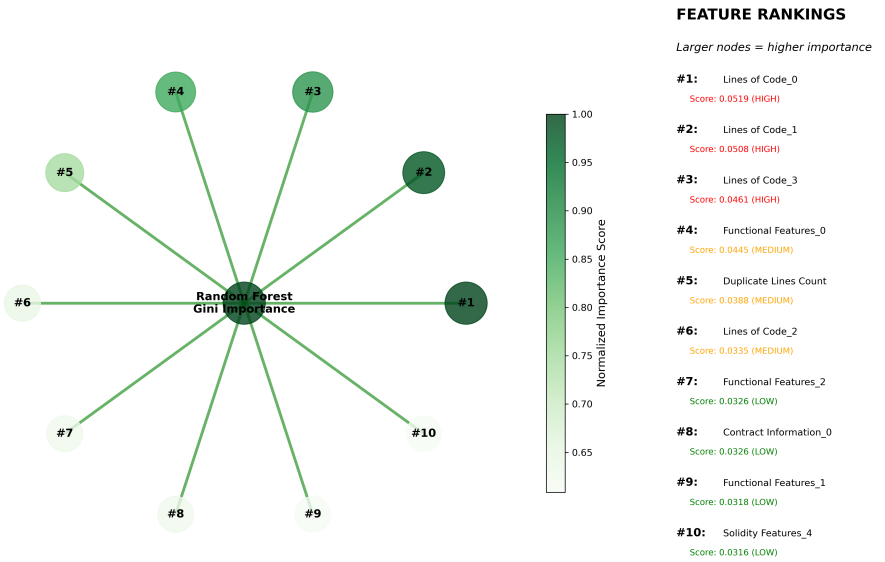

Fig. 3. Random forest Gini Importance Tree (Node size= Importance level)

XGBoost Gain Analysis: As shown in Figure 4, XGBoost gain puts strong weights on AST-related features, with `AST Features_ast_len_exportedSymbols` achieving the maximum gain score (148.14). The gradient boosting approach also valued AST Features `AST_len_nodes` (72.15) and `AST_nodetype_encoded` (100.97), highlighting the importance of syntactic structure in finding potential vulnerability

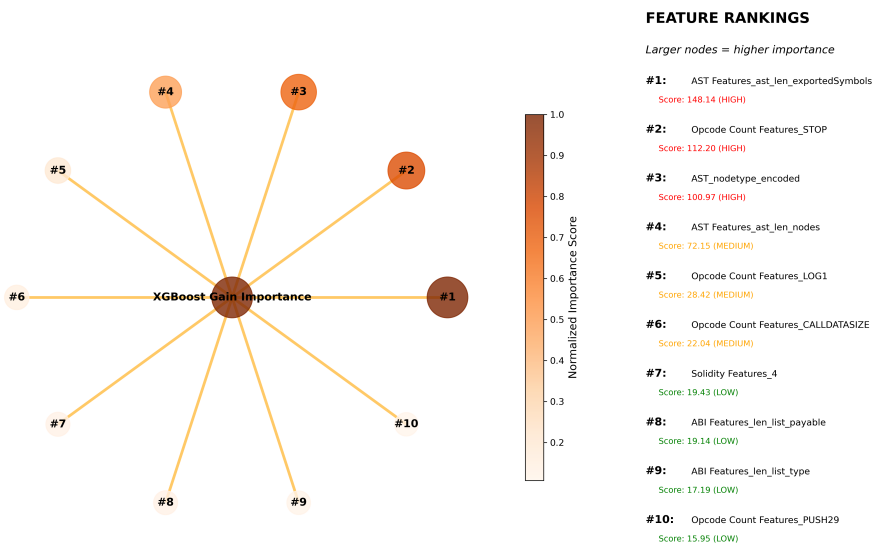

Fig. 4. XGBoost Gain Importance Tree (Node size= Importance level)

SHAP Value Interpretation: The SHAP analysis shown in Figure 5 provided model-agnostic insights, also suggesting `AST Features_ast_len_exportedSymbols` (0.3197) and `AST Features_ast_len_nodes` (0.2622) as the strongest contributors. Since SHAP values are based on the fundamentals of game-theory, the weight of these these features suggested that AST-based features are most important and play a central role in shaping predictions across the different vulnerability categories.

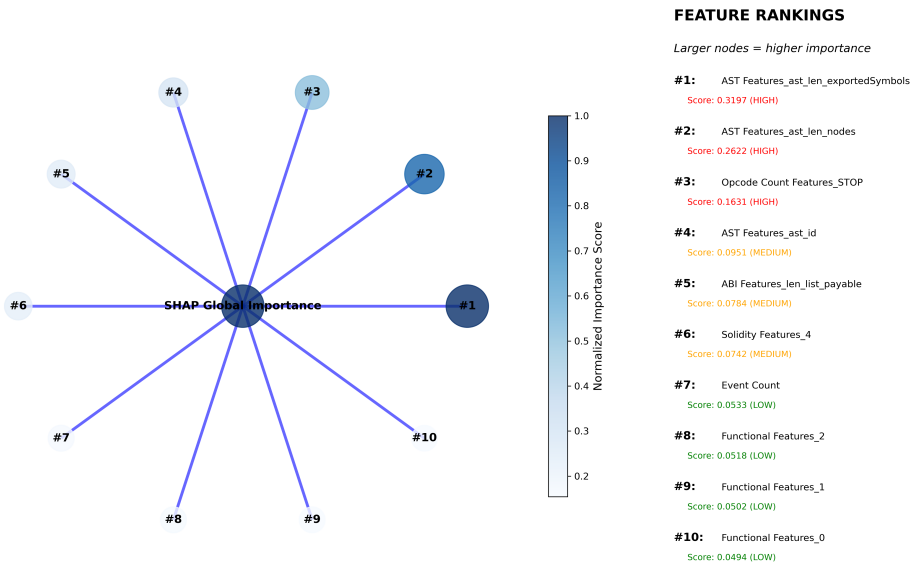

Fig. 5. SHAP Global Importance Tree (Node size= Importance level)

4.3 Feature Category Analysis

Figure 6 shows the distribution of the average importance in the feature categories for the top 20 features, revealing distinct patterns in vulnerability detection.

AST Features Dominance: Among the top 20 features, the abstract syntax tree features were the main part of the feature importance contribution. From Figure 6 it can be understood that the AST feature has the highest median importance and the larger range, suggesting that other AST features also have a great impact on decision making of machine learning models. Since AST also usually collects extra information about the program due to the consecutive stages of analysis by the compiler [21], which means that the syntax structure and program patterns are fundamental to attacks if overlooked.

Code Metrics Significance: Traditional code complexity measures had a major role in maintaining their importance, with a consistent median around 0.37. As the distribution is quite tight, it can be assumed that the different metrics of code contribute almost equally to the detection of vulnerabilities, which in turn supports the hypothesis that code complexity is related to security risk.

Functional Features : The presence of this category of features points at the influence of program behavior and interface design can lead to some vulnerability.

Other Categories: Opcode Features, ABI Features, Bytecode Features, Solidity Features and Other features like contract information contributed smaller,

but nevertheless, these categories of features indicate the patterns of low-level execution and specifications of interfaces used for comprehensive vulnerability detection provide additional information.

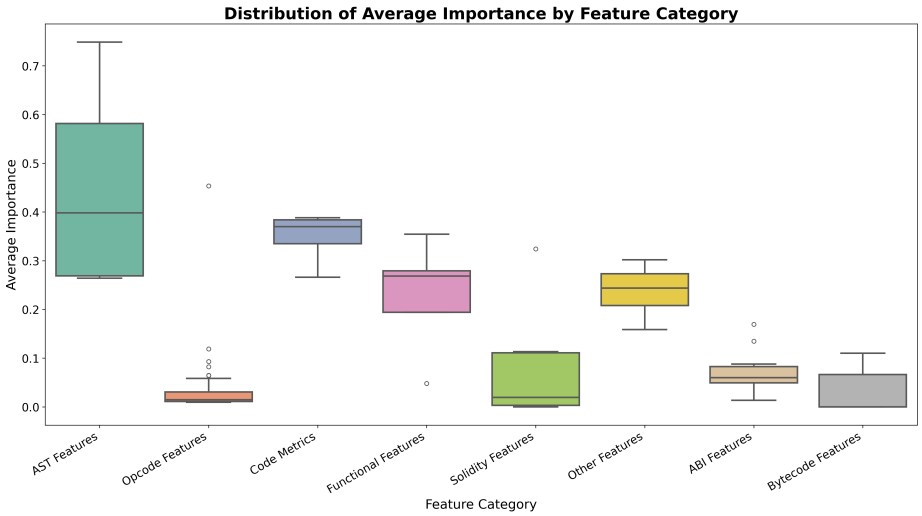

Fig. 6. Average importance by feature category top 20 features

4.4 Feature Ranking Stability Analysis

Figure 6 presents the change in the ranking behavior of the top 10 features in the different method methods.

Stable High-Performers: Notably, `AST Features_ast_len_exportedSymbols` and `AST Features_ast_len_nodes` maintain consistently high rankings across all methods, with low ranking variation. These features represent the most robust and method-independent importance signals.

Method-Dependent Features: `Opcode Count Features_STOP` shows eye catching ranking variation, appearing in low positions for Random Forest but achieving high importance in XGBoost and SHAP. This steep variation suggests that opcode-level features are especially valuable for gradient boosting approaches but play a smaller role in tree based ensemble models.

Code Metrics Consistency: The feature `Lines of Code` display consistent ranking stability, usually having slight mid-to-high positions across methods. This monotonous nature supports their role as reliable vulnerability indicators regardless of the analytical approach that has been used.

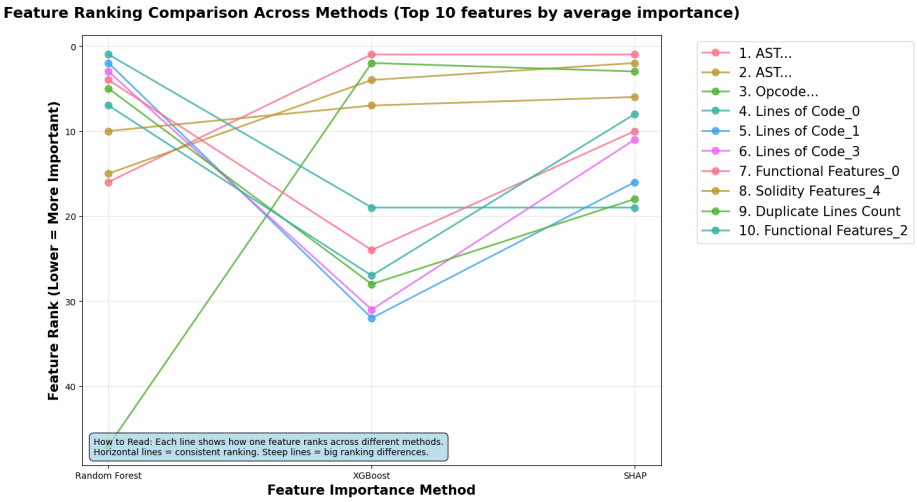

Fig. 7. Feature ranking comparison across methods (Top 10 features by average importance)

5 Conclusion and Future Work

With the introduction of smart contracts, traditional contracts have been transitioned into smart contracts, which need to be secure and reliable for everyone. Regarding the security of smart contracts, there is a lot of work which has developed high-security systems and machine learning tools for the detection of vulnerabilities For example Mythril [5] and Slither [6] . In addition, new datasets for example BCCC-CIC-Bell-DNS-2024 [30] have been created for the detection of more vulnerabilities and the creation of more advance models.

Our proposed approach conducted an analysis using multiple feature importance algorithms to reduce bias and to generate a generalize set of importance score to understand the features that are more important for machine learning algorithms in the classification of vulnerabilities.

The highest importance of AST Features, projects that smart contracts that have a complex structure are more prone to vulnerabilities. A proper security review should be done for the smart contracts codes that have a high complexity. Across the different importance model, the importance of lines of code suggest a positive correlation between code size and vulnerability. This supports the hypothesis and also minimizes the code complexity and reduces the total length of code.

As the major importance of features in the result is distributed among AST features, functional, opcode, bytecode categories, it suggests that vulnerability detection in smart contract requires features having all the information on code quality, logic, operations, and function types. Looking ahead, this work opens up several enhancements for further research. While this paper focuses on feature importance by encoding all the vulnerabilities into a single target column, there

is a potential to also look for feature importance by the different types of vulnerabilities. Additionally, there is a possibility to work for interactions between different types of features for a vulnerability class, thus providing more relevant, efficient scores and understanding on features for machine learning models.

Acknowledgements The author reflects his sincere thanks to the mentor for his insights, reviews, suggestions and directions of the work. Appreciations are owned to authors and creators of BCCC-SCsVuls-2024 dataset and its complete metadata and documentation. Gratitude is expressed to the open-source community for python, pandas, scikit learn, xgboost and shap. A generative AI assistant is used for grammar corrections. All content is created and verified by the author.

References

1. Safe Health Systems, "Safe Health Systems," *Safe Health Systems*, n.d., `https://safehealthsystems.com/`, accessed 7 October 2025.
2. Etherscan, "Deployed Contracts Chart," *Etherscan*, n.d., `https://etherscan.io/chart/deployed-contracts`, accessed [7 October 2025].
3. Cryptopedia Staff, "What Was the DAO Hack?," *Gemini Cryptopedia*, `https://www.gemini.com/cryptopedia/the-dao-hack-makerdao`, accessed 7 October 2025.
4. Santiago Palladino, "The Parity Wallet Hack Explained," *OpenZeppelin Blog*, 19 July 2017, `https://www.openzeppelin.com/news/on-the-parity-wallet-multisig-hack-405a8c12e8f7`, accessed 7 October 2025.
5. N. Sharma, S. Sharma, et al., A survey of Mythril, a smart contract security analysis tool for EVM bytecode, Indian J. Nat. Sci. 13 (75) (2023) 51003–51010.
6. Feist, Josselin and Grieco, Gustavo and Groce, Alex. "Slither: A Static Analysis Framework for Smart Contracts." 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), 2019, pp. 8-15. @INPROCEEDINGS8823898, author=Feist, Josselin and Grieco, Gustavo and Groce, Alex, booktitle=2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), title=Slither: A Static Analysis Framework for Smart Contracts, year=2019, volume=, number=, pages=8-15, keywords=Tools;Smart contracts;Static analysis;Optimization;Security;Detectors;Blockchain;Smart Contract;Solidity;Static Analysis;Vulnerability Detection, doi=10.1109/WETSEB.2019.00008
7. Sepideh HajiHosseinKhani, Arash Habibi Lashkari, Ali Mizani Oskui, Unveiling smart contract vulnerabilities: Toward profiling smart contract vulnerabilities using enhanced genetic algorithm and generating benchmark dataset, Blockchain: Research and Applications, Volume 6, Issue 2, 2025, 100253, ISSN 2096-7209, https://doi.org/10.1016/j.bcra.2024.100253. (https://www.sciencedirect.com/science/article/pii/S2096720924000666)
8. lundberg2017unifiedapproachinterpretingmodel, title=A Unified Approach to Interpreting Model Predictions, author=Scott Lundberg and Su-In Lee, year=2017, eprint=1705.07874, archivePrefix=arXiv, primaryClass=cs.AI, url=https://arxiv.org/abs/1705.07874,
9. Terrence Gatsby, "Implementing Blockchain in Telescope Networks," *Beyond the Hype*, Jun. 25, 2025. Available at: `https://beyondthehype.terrencegatsby.com/`

blockchain/implementing-blockchain-in-telescope-networks/ [Accessed: Oct. 9, 2025].
10. Investopedia Team, "Ethereum: What It Is, How It Works, and Who Created It," *Investopedia*, n.d., https://www.investopedia.com/terms/e/ethereum.asp, accessed [08 October 2025].
11. "The Role of Smart Contracts in Modern Business Transactions," *Medium/Coinmonks*, [Publication Date], https://medium.com/coinmonks/the-role-of-smart-contracts-in-modern-business-transactions-26007a47fe42, accessed [08 October 2025].
12. fonal2025dataset, title=Dataset of Yul Contracts to Support Solidity Compiler Research, author=Fonal, Krzysztof, journal=arXiv preprint arXiv:2506.19153, year=2025
13. Solidity Team, "Solidity 0.8.30 Documentation," *Solidity Official Documentation*, published 2024, https://docs.soliditylang.org/en/v0.8.30/, accessed 13 October 2025.
14. @ArticleBreiman2001, author=Breiman, Leo, title=Random Forests, journal=Machine Learning, year=2001, month=Oct, day=01, volume=45, number=1, pages=5-32, issn=1573-0565, doi=10.1023/A:1010933404324, url=https://doi.org/10.1023/A:1010933404324
15. Akshay Sharma, "Machine Learning 101: Decision Tree Algorithm for Classification," *Analytics Vidhya Blog*, published 1 March 2021, https://www.analyticsvidhya.com/blog/2021/02/machine-learning-101-decision-tree-algorithm-for-classification/, accessed 7 October 2025.
16. title = xgboost: Extreme Gradient Boosting, author = Tianqi Chen and Tong He and Michael Benesty and Vadim Khotilovich and Yuan Tang and Hyunsu Cho and Kailong Chen and Rory Mitchell and Ignacio Cano and Tianyi Zhou and Mu Li and Junyuan Xie and Min Lin and Yifeng Geng and Yutian Li and Jiaming Yuan and David Cortes, year = 2025, note = R package version 3.2.0.0, url = https://github.com/dmlc/xgboost,
17. @articleFENG2024100209, title = An interpretable model for large-scale smart contract vulnerability detection, journal = Blockchain: Research and Applications, volume = 5, number = 3, pages = 100209, year = 2024, issn = 2096-7209, doi = https://doi.org/10.1016/j.bcra.2024.100209, url = https://www.sciencedirect.com/science/article/pii/S2096720924000228, author = Xia Feng and Haiyang Liu and Liangmin Wang and Huijuan Zhu and Victor S. Sheng, keywords = Blockchain, Vulnerability detection, Smart contract,
18. Ethereum Gas Fees, https://ethos.dev/ethereum-gas-fees, Accessed: 15 August 2025
19. author = Huang, Yongfeng and Bian, Yiyang and Li, Renpu and Zhao, J. and Shi, Peizhong, year = 2019, month = 10, pages = 1-1, title = Smart Contract Security: A Software Lifecycle Perspective, volume = 7, journal = IEEE Access, doi = 10.1109/ACCESS.2019.2946988
20. C. W. Fraser and D. R. Hanson, *A Retargetable C Compiler: Design and Implementation*, 1st ed., Addison-Wesley Professional, Boston, MA, 1995.
21. Wikimedia Foundation, "Abstract syntax tree," *Wikipedia*, https://en.wikipedia.org/wiki/Abstract_syntax_tree, Accessed Oct. 9 2025.
22. S. Sen, "What is an ABI?", QuickNode Guides, Jul. 9, 2025. Available at: https://www.quicknode.com/guides/ethereum-development/smart-contracts/what-is-an-abi [Accessed: Oct. 9, 2025]
23. R. Sheldon, "What is bytecode?", TechTarget, Jun. 15, 2022. Available at: https://www.techtarget.com/whatis/definition/bytecode [Accessed: Oct. 9, 2025]

24. Z. Pratap, "What Are ABI and Bytecode in Solidity?", Chainlink Blog, Aug. 17, 2022. Available at: https://blog.chain.link/what-are-abi-and-bytecode-in-solidity/ [Accessed: Oct. 9, 2025].
25. "Opcode," *Wikipedia*, Available at: https://en.wikipedia.org/wiki/Opcode [Accessed: Oct. 9, 2025].
26. wolflo, "evm-opcodes: A quick reference for EVM opcodes," GitHub repository, Available at: https://github.com/wolflo/evm-opcodes [Accessed: Oct. 9, 2025].
27. article, author = Strobl, Carolin and Boulesteix, Anne-Laure and Zeileis, Achim and Hothorn, Torsten, year = 2007, month = 02, pages = 25, title = Bias in Random Forest Variable Importance Measures: Illustrations, Sources and a Solution." BMC Bioinformatics, 8(1), 25, volume = 8, journal = BMC bioinformatics, doi = 10.1186/1471-2105-8-25
28. Takefuji2025, author=Takefuji, Yoshiyasu, title=Addressing feature importance biases in machine learning models for early diagnosis of type 1 Gaucher disease, journal=Journal of Clinical Epidemiology, year=2025, month=Feb, day=01, publisher=Elsevier, volume=178, issn=0895-4356, doi=10.1016/j.jclinepi.2024.111619, url=https://doi.org/10.1016/j.jclinepi.2024.111619
29. Bilodeau2024, title=Impossibility theorems for feature attribution, volume=121, ISSN=1091-6490, url=http://dx.doi.org/10.1073/pnas.2304406120, DOI=10.1073/pnas.2304406120, number=2, journal=Proceedings of the National Academy of Sciences, publisher=Proceedings of the National Academy of Sciences, author=Bilodeau, Blair and Jaques, Natasha and Koh, Pang Wei and Kim, Been, year=2024, month=jan
30. SHAFI2024109436, title = Unveiling malicious DNS behavior profiling and generating benchmark dataset through application layer traffic analysis, journal = Computers and Electrical Engineering, volume = 118, pages = 109436, year = 2024, issn = 0045-7906, doi = https://doi.org/10.1016/j.compeleceng.2024.109436, url = https://www.sciencedirect.com/science/article/pii/S0045790624003641, author = MohammadMoein Shafi and Arash Habibi Lashkari and Hardhik Mohanty, keywords = DNS analysis, Behavioral profiling, Application layer security, Anomaly detection, Malicious behavior identification, Pattern extraction, Behavior similarity calculation, ALFlowLyzer, BCCC-CIC-bell-DNS-2024,