An Analysis of Syntactic Evolution and Detection Performance in Smart Ponzi Schemes

Rikuto Otsu[1], Yin Minn Pa Pa[2], Katsunari Yoshioka[2], and Kazumasa Omote[1]

[1] University of Tsukuba, Tsukuba, Japan
s2420528@u.tsukuba.ac.jp
[2] Yokohama National University, Yokohama, Japan

Abstract. Smart Ponzi Schemes (SPS), which are deployed on smart contracts, have undergone increasing syntactic evolution over the years, raising concerns about the declining effectiveness of conventional detection methods. Even with PonziGuard, a recent high-performance detection system based on runtime behavior analysis, there has been limited investigation into how syntactic changes affect detection performance. This study analyzes SPS contracts detected by PonziGuard and conducts syntactic clustering based on opcode sequences, along with semantic visualization using Word2Vec and PCA, to examine structural trends over time. We further perform time-series classification experiments using multiple machine learning models and feature representations to evaluate the impact of these syntactic differences on detection accuracy. Our results reveal a significant drop in performance, especially against obfuscated and delegate-call-based contracts, when early-trained models are applied to newer samples. These findings highlight the limitations of conventional detection approaches and emphasize the need for adaptive systems that can respond effectively to evolving syntactic patterns in malicious smart contracts.

Keywords: Smart contract · Ponzi scheme · Machine Learning · Fisher's exact test

1 Introduction

With the advancement and widespread adoption of blockchain technology in recent years, its applications have expanded across various domains such as finance and healthcare. Blockchain is a decentralized system that ensures transparency and tamper resistance of transactional data. By enabling secure and efficient transactions without intermediaries, it has attracted significant attention across many industries. In particular, decentralized finance (DeFi), token issuance, and voting systems are being actively developed using blockchain-based applications. Among them, smart contracts play a central role and are expected to be widely adopted in various fields.

However, the proliferation of blockchain and smart contracts has also raised concerns about the increasing abuse of their anonymity and immutability for

fraudulent activities. One representative example is the emergence of Smart Ponzi Schemes (SPS). SPS are automated fraud mechanisms built on smart contracts that promise high returns to investors, while in reality distributing the funds collected from new participants to earlier investors. As these schemes scale, the resulting financial damage becomes increasingly severe, necessitating early detection and prevention.

In recent years, machine learning-based approaches have been introduced to detect SPS, showing promising results in pattern recognition and anomaly detection compared to traditional rule-based methods. One of the most effective methods to date is PonziGuard, proposed by Liang et al. [10], which outperforms prior techniques that rely on opcode frequency or transaction history. PonziGuard demonstrated high detection performance and practical efficacy by identifying 805 SPS on the Ethereum Mainnet.

Despite its performance, several issues remain. First, the dataset used in experiments may not fully reflect recent fraud trends, especially in terms of syntactic diversity and obfuscation. Second, there is a lack of quantitative evaluation regarding missed detections, limiting the understanding of detection coverage. Third, it is unclear whether the observed changes in detected SPS reflect a decline in SPS themselves or the limitations of the detector in capturing newer variants.

This study aims to analyze the syntactic evolution of SPS and its impact on detection performance to reveal the practical limits of PonziGuard and identify future directions for improving detection models. To this end, we pose the following two research questions:

RQ1: Have the syntactic characteristics of SPS contracts changed over time?
RQ2: How does this syntactic evolution affect the performance of detection models?

To address these questions, we perform a temporal analysis of opcode-based syntactic patterns of SPS contracts detected by PonziGuard. Furthermore, we evaluate the effect of these changes on detection performance by comparing multiple classification models trained and tested on different time periods. The results provide insights into how syntactic diversity influences detection and underscore the need for continuous model updates in the design of SPS detection systems.

2 Preliminaries

2.1 Smart Contracts

Smart contracts are programs that automatically execute agreements or transactions when predefined conditions are met, and they primarily operate on blockchain platforms such as Ethereum. The Ethereum Virtual Machine (EVM) serves as the execution environment for smart contracts, providing a foundation that allows developers to build secure and efficient code.

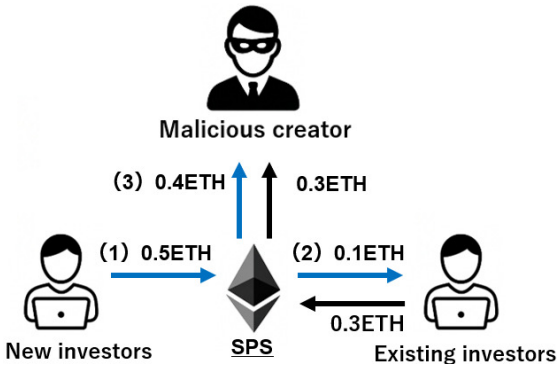

Fig. 1. Example flow of a smart Ponzi scheme

The primary advantage of smart contracts lies in eliminating the need for intermediaries or third-party involvement, which reduces transaction costs and enhances transparency. Additionally, once deployed on the blockchain, smart contracts cannot be altered or deleted, granting them a high degree of tamper resistance. This immutability further enhances their reliability.

2.2 Smart Ponzi Scheme

Smart Ponzi schemes are a type of fraud discovered on Ethereum, in which traditional Ponzi schemes are automated using smart contracts. A Ponzi scheme is a fraudulent structure that promises high returns to attract new investors and redistributes the funds collected from them to earlier investors. When new participation ceases, the scheme collapses due to a lack of funds.

As illustrated in Fig. 1, in a smart Ponzi scheme, users send funds to a contract disguised as an investment program. A portion of the funds is redistributed to previous participants, while the majority is siphoned off by the malicious creator. By leveraging smart contracts, this process is fully automated and executed transparently on the tamper-resistant blockchain.

However, the anonymity and immutability of blockchain can facilitate the concealment of fraudulent behavior and hinder recovery from damage. Moreover, the fact that the contract code is publicly available may lead to a false sense of trust, causing users to participate without fully understanding the mechanism.

Due to these factors, smart Ponzi schemes pose more serious risks than traditional Ponzi schemes, particularly in terms of irreversibility and the scale of potential damage. Bartoletti et al.[2] conducted a comprehensive study on Ponzi schemes on Ethereum, revealing various implementation patterns and highlighting the long-term economic impact of such schemes.

Recently, research has been progressing on detection methods using machine learning and other analytical techniques to enable early identification of these fraudulent contracts.

3 Related Work

In the field of smart Ponzi scheme detection using machine learning, several approaches have been proposed that leverage transaction data, code features, and runtime behavior. This section introduces representative studies for each category.

3.1 Detection Methods Using Transaction Data

In the study by Hu et al. [9], over 10,000 transaction records of Ethereum smart contracts were analyzed, and 14 behavioral features were extracted to classify contracts using an LSTM model. Their method demonstrated high accuracy in identifying contracts related to games, finance, and gambling, as well as detecting malicious contracts exhibiting abnormal behavior.

Compared to the Ponzi-scheme-specific method proposed by Chen et al. [4], Hu's method shows lower accuracy due to the absence of opcode-level analysis. However, it offers higher generalizability and is applicable beyond fraud detection, making it a versatile framework for broader contract classification tasks.

3.2 Detection Methods Based on Code Information

Detection methods that utilize code-level features, particularly the frequency distribution of opcodes, have demonstrated high effectiveness in many studies [4], [13], [6], [14]. Wang et al. [13] proposed PSD-OL (Ponzi Scheme Detection via Oversampling-based Long Short-Term Memory), which inputs opcode sequences and account-level information into an LSTM model, while applying SMOTE (Synthetic Minority Over-sampling Technique) [7] to address data imbalance. By capturing distinctive characteristics of SPS such as the frequent use of JUMP instructions, their approach achieved higher detection accuracy compared to prior methods. However, handling evolving fraud tactics (concept drift) remains a challenge.

Zheng et al. [14] proposed MulCas (Multi-view Cascade Ensemble), which does not rely on transaction information but instead uses bytecode and developer metadata of smart contracts. A key advantage of MulCas is its ability to detect potentially fraudulent contracts at deployment time, before any malicious behavior occurs—enabling proactive mitigation. In addition to opcode frequencies and N-gram patterns, it introduces a novel feature called "developer vectors," demonstrating its effectiveness. Furthermore, MulCas outperforms existing approaches in terms of recall and F1-score, particularly under imbalanced data conditions.

On the other hand, it faces challenges such as vulnerability to adversarial attacks and the use of outdated datasets like XBlock [1]. Improving the model's robustness and continuously updating the dataset are identified as important future tasks.

3.3 Detection Methods Based on Runtime Information

Liang et al. [10] proposed PonziGuard, a detection method that leverages runtime behavior information. A key feature of their approach is the construction of a Contract Runtime Behavior Graph (CRBG), which combines multiple runtime attributes such as stack operations, fund flows, and inter-function calls. Each function's opcode sequence is first embedded using Doc2Vec, and the resulting CRBG is then used as input to a Graph Neural Network (GNN) for classification.

In their experiments, PonziGuard outperformed all existing methods across various evaluation metrics and successfully detected 805 new Ponzi schemes on the Ethereum Mainnet. Manual verification of 50 contracts with available source code confirmed all to be fraudulent, demonstrating the method's effectiveness. The estimated total financial damage caused by these schemes exceeded 281,700 ETH.

Although the overhead of extracting runtime information—including modifications to the EVM—is considered acceptable, the full process, which involves fuzzing and graph construction, is computationally intensive. This raises scalability concerns, particularly in the context of full-scale scanning or real-time detection applications.

3.4 Limitations of Existing Methods

Through related studies, several challenges have been identified in existing research on smart Ponzi scheme detection.

First, the datasets used in many studies are outdated. Popular datasets such as XBlock [1] are based on information from before 2019, which may not adequately reflect recent trends such as diversification and obfuscation in scheme designs. Additionally, imbalances in the number of contracts across different years raise concerns about the reliability of temporal evaluations. To address these issues, it is necessary to construct up-to-date and comprehensive datasets, ensure continuous data updates, and improve the adaptability and performance of detection models.

Second, class imbalance remains a critical issue affecting detection performance. Labeled Ponzi scheme data are still scarce, leading to risks of overfitting and poor generalization. Oversampling techniques such as SMOTE and the development of robust models—like the one proposed by Wang et al. [12]—that are less sensitive to minority class imbalance are essential.

Finally, vulnerability to adversarial attacks is another key concern. Detection methods that rely heavily on code features are particularly susceptible to attacks involving the insertion of meaningless opcodes (adversarial examples), which can significantly degrade detection accuracy.

4 Limitations of PonziGuard

As discussed in the previous section, the study by Liang et al. [10] proposed PonziGuard, a runtime-based detection method that demonstrated high per-

formance both on ground-truth datasets and on Ethereum Mainnet. However, several limitations remain, as outlined below.

- **Lack of quantitative analysis on syntactic trends:** While PonziGuard provides a brief mention of the temporal distribution of detected SPS cases, it does not quantitatively analyze syntactic features or their evolution over time. Although correlations between SPS trends and social events such as the COVID-19 pandemic are suggested, no causal or quantitative relationship is established between such external factors and contract design patterns.
- **Limited evaluation of detection performance:** Although 805 SPS contracts were detected on Mainnet and 50 were manually verified as true positives, no recall, F1-score, or other quantitative metrics were reported to assess the rate of false negatives. In particular, the risk of misclassification due to syntactic drift remains unaddressed, leaving the robustness of the model underexplored.

These limitations highlight the need for a more careful assessment of model adaptability to recent trends and structural variations in contract design, as well as for the use of comprehensive evaluation metrics.

Accordingly, in proposing a new detection approach, it is essential to first identify the syntactic characteristics of smart Ponzi schemes and then clarify the relationship between their evolution and detection performance. To that end, RQ1 aims to quantitatively analyze changes in syntactic trends over time, while RQ2 evaluates how such structural evolution affects the detection performance of classification models.

5 RQ1: Time-Series Analysis of Syntactic Features in SPS Contracts

In the detection of smart Ponzi schemes (SPS), syntactic evolution—such as increasing structural diversity and obfuscation—has become a major challenge for existing detectors. Particularly, it is plausible that SPS developers intentionally modify contract structures to evade detection. Understanding such syntactic changes over time is essential for identifying the limitations of current detectors and for guiding the design of future detection systems.

This section investigates temporal trends in syntactic patterns through clustering and statistical analysis of opcode sequences, focusing on SPS contracts detected by PonziGuard [11].

5.1 Dataset

Following the methodology of Liang et al., we utilize 805 SPS contract addresses made publicly available by PonziGuard on GitHub [11]. For each address, we collect deployment timestamps and code information using the Etherscan API [5], and format the data for analysis.

Note that source code and bytecode are not publicly available for certain contracts on Etherscan. We exclude 63 such contracts, resulting in 742 SPS contracts as our final dataset. The bytecode of each contract is converted into opcode sequences, which serve as the basis for extracting syntactic features.

5.2 Methodology

To address RQ1, we perform syntactic clustering of smart Ponzi schemes (SPS) based on opcode sequences, followed by extraction of characteristic opcodes for each cluster. The analysis consists of the following three steps.

Step 1: Vectorization and Clustering The 742 SPS contracts collected in the previous section are treated as documents, and their opcode sequences are vectorized using TF-IDF to capture their opcode frequency patterns in a quantitative manner.

Subsequently, k-means clustering is applied to group syntactically similar contracts. The number of clusters k is determined based on the change in the Silhouette score, with $k = 5$ selected as the optimal value in this study.

To evaluate temporal trends, the distribution of clusters is analyzed across different years.

Step 2: Quantitative Extraction of Cluster-Specific Opcodes To quantitatively characterize each cluster C_i, we compare the contracts in C_i with those in its complement set $\neg C_i$ and compute the occurrence rate difference $\Delta(w, C_i)$ for each opcode w[3]. The difference is defined as:

$$\Delta(w, C_i) = \frac{n_{C_i}(w)}{N_{C_i}} - \frac{n_{\neg C_i}(w)}{N_{\neg C_i}} \tag{1}$$

where $n_{C_i}(w)$ is the number of contracts in cluster C_i that contain opcode w, and N_{C_i} is the total number of contracts in C_i. Similarly, $n_{\neg C_i}(w)$ and $N_{\neg C_i}$ refer to the corresponding values for the complement set $\neg C_i$.

To assess statistical significance, we construct a 2×2 contingency table for each opcode and perform Fisher's exact test[4] [8]. To correct for multiple comparisons across opcodes, we apply false discovery rate (FDR) correction using the Benjamini–Hochberg method [3]. Opcodes with adjusted p-values below 0.05 are considered statistically significant.

This process is repeated for each cluster, and the top 10 opcodes with the highest $\Delta(w, C_i)$ and statistically significant p-values are extracted as representative syntactic features.

[3] This value indicates how much more frequently an opcode appears within a specific cluster compared to outside of it. In this study, opcodes with high $\Delta(w, C_i)$ are treated as representative keywords for the cluster.

[4] Fisher's exact test is a statistical method for determining whether two categorical variables are significantly associated. In our context, it tests whether the presence of an opcode differs significantly between clusters.

We note that relying solely on raw frequency or TF-IDF values often results in frequent but generic opcodes being ranked highest, potentially obscuring inter-cluster differences. Therefore, our method focuses on opcodes that are particularly overrepresented within each cluster, allowing for clearer syntactic differentiation among them.

Step 3: Visualization of the Semantic Space of Syntax To visually assess the semantic similarity between clusters, we first vectorize each opcode using Word2Vec and compute the average vector for each contract, thereby constructing a semantic vector representation at the contract level. We then apply Principal Component Analysis (PCA) to reduce the dimensionality to two dimensions and plot the semantic centroids of the clusters on a 2D plane.

This visualization enables us to identify the semantic distances and similarities between clusters, as well as to detect clusters that exhibit structurally divergent patterns. Consequently, it allows us to capture not only the frequency trends of syntactic features but also their semantic evolution and divergence over time.

5.3 Results

This section presents a temporal evaluation of syntactic features based on the clustering and statistical analysis methods described earlier. We focus on the following three aspects: (1) temporal shifts in cluster composition ratios of SPS contracts, (2) statistically significant representative opcodes for each cluster, and (3) visualization of inter-cluster similarity in semantic space.

Temporal Shifts in Cluster Composition Fig. 2 shows the cluster composition ratios of detected SPS contracts for each year from 2015-2016 to 2020. In 2015-2016, Cluster 2 accounted for more than half of the contracts, indicating that early SPS contracts were concentrated around a specific syntactic pattern.

In contrast, by 2017, the cluster composition had diversified, with increases in the proportions of Cluster 0 (53.9%), Cluster 1 (13.9%), and Cluster 3 (18.9%), suggesting a structural shift.

In 2018, Cluster 3 (44.1%) and Cluster 4 (40.2%) became dominant, and by 2020, all detected SPS contracts fell under Cluster 4, indicating a trend toward reduced syntactic diversity over time.

Notably, both Cluster 1 and Cluster 3 disappeared entirely after 2019. These results suggest an evolution in syntactic structure over time, potentially due to detection limitations or evasive adaptations by developers.

Representative Opcodes by Cluster Table 1 shows representative opcodes in each cluster, selected based on the highest frequency gap $\Delta(w, C_i)$ and statistical significance ($p < 0.05$) confirmed by Fisher's exact test. Five distinctive opcodes were listed for each cluster.

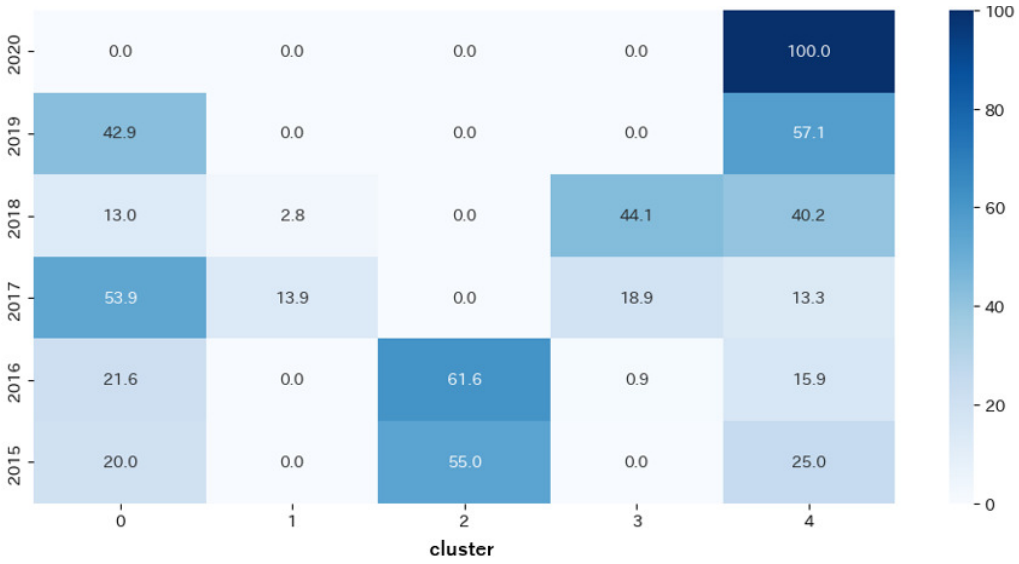

Fig. 2. Yearly composition ratio by cluster (2015–2020)

Based on these representative instructions, the following syntactic tendencies were observed in each cluster:

- **Cluster 0 (State-checking type)**: This cluster is characterized by a frequent use of `SWAP` and `DUP`, along with external state retrieval instructions such as `BALANCE` and `TIMESTAMP`. These contracts focus on validating transfer conditions based on environmental data.
- **Cluster 1 (Obfuscated and delegate-call-based type)**: Featuring `DELEGATECALL` and undefined instructions like `UNKNOWN_0xd0`, this cluster reflects structures designed to be opaque and difficult to analyze, often employing obfuscation or delegated execution.
- **Cluster 2 (Gas control and initialization type)**: This cluster includes instructions like `CODECOPY`, `GASLIMIT`, and `CODESIZE`, suggesting usage in initialization code that configures contract behavior dynamically at deployment time.
- **Cluster 3 (Exception handling and disguise type)**: Highlighted by opcodes such as `REVERT`, `LOG1`, and `GAS`, these contracts may simulate refund handling or abnormal termination to deceive users.
- **Cluster 4 (Control-structure type)**: Dominated by basic stack manipulation instructions like `DUP` and `PUSH`, this cluster forms abstract, reusable control flows such as loops and conditional branches. These structures are syntactically simple yet adaptable across diverse scenarios.

These clusters reveal structural biases that are difficult to identify using frequency-based analysis alone. When combined with temporal distribution pat-

Table 1. Representative opcodes in each cluster with statistically significant differences

Cluster	Opcode	Frequency Gap	p-value
0	SWAP6	0.470	< 0.001
	DUP14	0.357	< 0.001
	SWAP12	0.331	< 0.001
	TIMESTAMP	0.315	< 0.001
	LOG1	0.287	< 0.001
1	DELEGATECALL	0.970	< 0.001
	INVALID_0x6c	0.830	< 0.001
	UNKNOWN_0xd0	0.811	< 0.001
	PUSH19	0.772	< 0.001
	CALLDATACOPY	0.677	< 0.001
2	UNKNOWN_0xa9	0.522	< 0.001
	UNKNOWN_0xd	0.517	< 0.001
	UNKNOWN_0xec	0.514	< 0.001
	UNKNOWN_0xd9	0.509	< 0.001
	INVALID_0x64	0.492	< 0.001
3	PUSH29	0.550	< 0.001
	REVERT	0.536	< 0.001
	UNKNOWN_0xfe	0.474	< 0.001
	PUSH6	0.413	< 0.001
	LOG1	0.353	< 0.001
4	DUP10	0.373	< 0.001
	UNKNOWN_0xfe	0.335	< 0.001
	REVERT	0.333	< 0.001
	DUP7	0.308	< 0.001
	DUP8	0.306	< 0.001

terns, they provide useful insights for evaluating the limitations and vulnerabilities of existing detection systems.

Cluster Distribution in Semantic Space Fig. 3 visualizes the semantic distances between cluster centroids based on opcode usage. Each contract was encoded as a vector by averaging Word2Vec embeddings of its opcodes, and the resulting cluster-level centroids were projected onto a 2D space using Principal Component Analysis (PCA).

The plot shows that Cluster 1 and Cluster 3 are distributed far from the others, indicating that their syntactic patterns are semantically distinct. This supports the hypothesis that obfuscated or misleading opcode structures—such as those involving delegate calls or fake error handling—occupy outlier regions in the semantic space and are potentially harder for detection models to handle.

In contrast, Clusters 0, 2, and 4 are located relatively close to each other, implying a higher degree of semantic similarity. These clusters may represent more conventional or general-purpose syntactic patterns, to which existing detectors may be more robust.

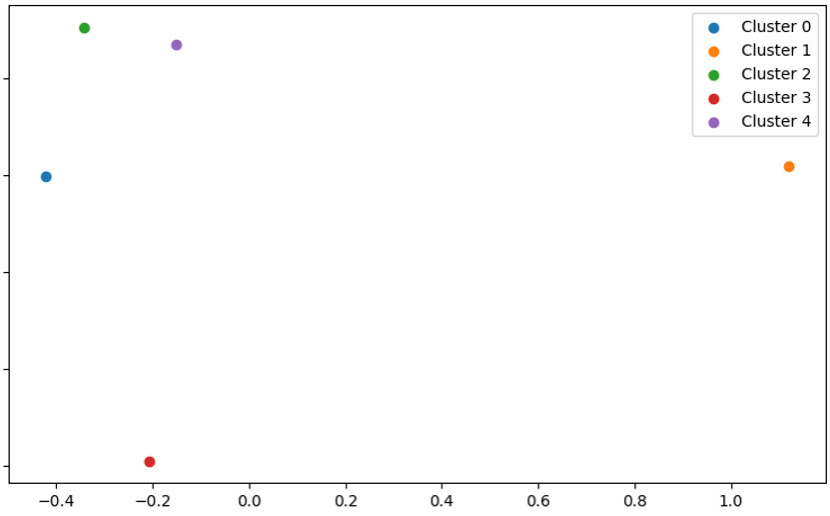

Fig. 3. Cluster centroids in semantic space (reduced to 2D via PCA)

6 RQ2: Detection Performance with Respect to Syntactic Features of SPS Contracts

The previous section demonstrated that the syntactic patterns of Smart Ponzi Schemes (SPS) have evolved over time. In this section, our objective is to evaluate how these changes affect detection performance. We conduct a series of classification experiments and analyze misclassification patterns to quantitatively assess the relationship between syntactic evolution and model performance, such as missed detections or accuracy shifts.

6.1 Dataset

As in RQ1, we use 742 SPS contracts detected by PonziGuard. These are divided into three temporal groups based on their deployment timestamps: 2015–2016, 2017–2018, and 2019–2020.

To evaluate the classification performance of the detection models, we prepare a set of non-SPS contracts as a control group. These contracts are randomly sampled using the Etherscan API [5], ensuring that the number of non-SPS contracts matches that of SPS contracts in each time period to avoid data imbalance.

All contracts are converted into opcode sequences, which are then used for feature extraction, training, and evaluation in the classification models. This setup enables a quantitative evaluation of how classification performance based on syntactic features changes in response to the structural evolution of SPS contracts.

6.2 Methodology

To address RQ2, we conduct classification experiments to quantitatively evaluate the impact of syntactic evolution on detection performance. In particular, we focus on the tendency of existing detectors to produce false negatives (FN) as a result of structural changes. The analysis procedure consists of the following steps:

Step 1: Cross-Temporal Evaluation of Classification Performance To explicitly assess how changes in syntactic patterns affect classification accuracy, we fix the training period to 2015–2016 and use test sets from three distinct time periods: 2015–2016, 2017–2018, and 2019–2020. This setup allows us to compare the generalization ability of classifiers across different temporal contexts and measure the impact of syntactic evolution.

We adopt five widely used classification models from prior studies: Ridge, Support Vector Machine (SVM), Random Forest, XGBoost, and LightGBM. For input features, we use multiple representations derived from opcode sequences: TF-IDF, term count, bigram, bigram-based TF-IDF, and Word2Vec embeddings. By evaluating the F1 scores for each model-feature combination across different time periods, we can quantitatively analyze the variation in detection performance caused by syntactic evolution.

Step 2: Error Analysis of False Negatives When classification performance deteriorates, we analyze whether false negatives (FN) are disproportionately concentrated in specific syntactic clusters. This analysis leverages the clustering results obtained in RQ1.

We focus specifically on false negatives because our clustering and feature extraction processes are based solely on SPS contracts. Therefore, only false negatives—i.e., undetected SPS—can be analyzed in terms of syntactic characteristics. Furthermore, unlike false positives, which may cause false alarms, false negatives pose a higher real-world risk due to undetected fraud, making them a particularly critical metric.

In this step, we concentrate on the classifier that exhibited the largest variation in F1 score (XGBoost combined with TF-IDF in our case). We analyze the false negative samples from the 2017–2018 test set, and aggregate their distribution across syntactic clusters.

This analysis seeks to answer the following questions:

- Are detection failures concentrated in specific clusters?
- Do cluster-specific syntactic characteristics correlate with misclassification tendencies?

Through this analysis, our objective is to clarify whether structurally atypical SPS patterns are more prone to being overlooked by detection models, highlighting challenges for the design of more robust detectors.

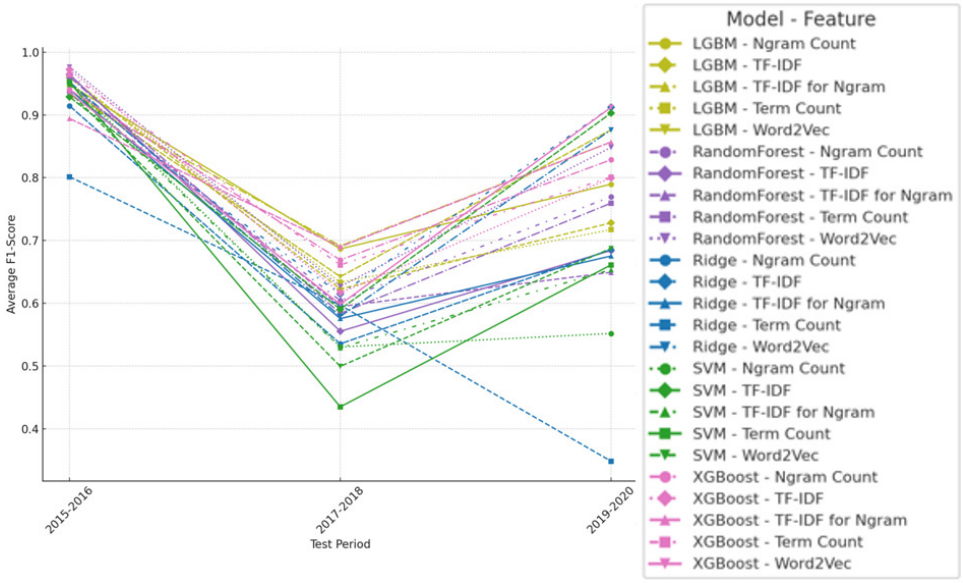

Fig. 4. F1 scores by model and feature type across time periods

6.3 Results

Performance Comparison Across Time Periods As shown in Fig. 4, all classification models trained on SPS contracts from 2015–2016 exhibited a performance drop of over 20 percentage points in F1 score when tested on the 2017–2018 dataset. Notably, the F1 scores showed partial recovery for the 2019–2020 period. This trend was consistent across all model-feature combinations, suggesting that syntactic evolution of SPS contracts significantly affects detection performance.

Cluster Distribution of False Negatives Next, we examine the cluster distribution of false negatives (FN) for the XGBoost + TF-IDF model, which showed the most significant performance variation. Table 2 shows the results for the 2017–2018 period. Notably, 67.2% of misclassified SPS contracts fell into Cluster 1 (Obfuscated / Delegate-call Type), followed by 20.8% in Cluster 0 (State-check Type). These clusters were either absent or structurally different from those seen in the 2015–2016 training data, suggesting that the model struggled to generalize to these previously unseen patterns.

These results quantitatively indicate that the syntactic evolution of SPS contracts—particularly the emergence of structurally distinct patterns such as Cluster 1 in 2017–2018—contributed to a temporary decline in detection performance, highlighting the limitations of existing detectors.

Table 2. Cluster distribution of false negatives (FN) in 2017–2018

Cluster	FN Ratio
Cluster 1	67.2%
Cluster 0	20.8%
Cluster 4	9.5%
Cluster 3	2.4%
Cluster 2	0.0%

7 Discussion

7.1 Potential Evasion Strategies and Evolution in SPS Design

The temporal changes in syntactic patterns identified in RQ1 suggest that the designers of smart Ponzi schemes (SPS) may have deliberately modified contract structures to evade detection. In particular, the sharp increase in Cluster 1 and Cluster 3 during 2017–2018 represents a significant departure from the structural patterns previously learned by standard detectors, implying a possible evasion strategy.

After 2019, we observed a clear convergence into Cluster 4 , which allows for multiple interpretations. One possibility is that detectors gradually adapted to previously evasive structures, prompting malicious actors to abandon those techniques in favor of simpler forms like Cluster 4. Another possibility is that obfuscated patterns remain difficult to detect, and current detectors are only effective at identifying syntactically simpler SPS contracts—leading to an apparent, but misleading, concentration in Cluster 4.

The former scenario reflects progress in detection capability, while the latter exposes the limitations of current detection methods and raises concerns about undetected threats. If the latter holds true, it underscores the urgent need for detection systems that are robust against evolving and obfuscated contract structures.

7.2 Limitations of PonziGuard and Relationship with Syntactic Trends

This study analyzed the relationship between syntactic trends and detection performance, focusing on SPS contracts detected by PonziGuard, to clarify the limitations of its adaptability.

In RQ2, classification models trained on SPS contracts from 2015–2016 exhibited a substantial drop in F1 score when applied to data from 2017–2018, with misclassifications (FN) highly concentrated in Cluster 1 . This result suggests that the emergence of previously unseen syntactic patterns significantly impaired the model's generalization ability.

In contrast, the F1 scores partially recovered for the 2019–2020 dataset. This improvement is likely due to the re-dominance of Cluster 4, which shares greater

syntactic similarity with the original training data, enabling classifiers to perform better again.

Overall, these findings reveal that detection performance is highly dependent on specific syntactic patterns and vulnerable to structural evolution. When new syntactic forms appear that were not present in the training data, generalization deteriorates, resulting in concentrated detection failures (FN).

These results highlight inherent limitations in detecting syntactically diverse or evolving SPS contracts. To ensure sustainable and robust fraud detection in smart contract environments, it is essential to design detectors that can adapt to evolving syntactic structures. Continuous model updates and improved resilience to syntactic evolution will be critical for maintaining detection efficacy over time.

7.3 Limitations

This study has several limitations. First, the analysis is limited to SPS detected by PonziGuard, excluding undetected instances. Therefore, the syntactic trends and temporal changes discussed in this study are based only on detectable SPS and may not reflect the overall landscape. Furthermore, since we relied on PonziGuard's detection results for labeling, not all analyzed contracts are guaranteed to be true SPS. In particular, many contracts lack publicly available source code, making it difficult to directly verify their execution logic or intended design.

Second, our analysis is limited to 742 contracts for which bytecode was successfully obtained. Sixty-three contracts that could not be retrieved were excluded. If these excluded contracts contain unique syntactic patterns, they are not captured in our findings. However, the excluded cases are not heavily biased toward specific years, and thus their impact on our year-based evaluation is considered limited.

Third, our syntactic analysis is based solely on opcode sequences, which do not directly reveal the developer's intent or the fraudulent nature of each cluster. This limitation is particularly relevant for obfuscated clusters such as Cluster 1, where source code is unavailable, making precise behavioral interpretation more challenging.

Fourth, contract deployment timestamps were obtained via the Etherscan API, which may include some degree of temporal uncertainty. Cases involving redeployment or proxy usage may particularly affect the accuracy of deployment timing. Nonetheless, such limitations are common in prior studies, and our approach is not significantly inferior in this regard.

In summary, this study analyzes only the syntactic tendencies of already detected SPS contracts and does not achieve a comprehensive understanding of syntactic diversity or evasion strategies. Future work should aim to include undetected SPS, analyze the full structural landscape, and conduct in-depth examinations where partial source code disclosure is available.

7.4 Future Work

This study analyzed the relationship between syntactic evolution and detection performance of SPS, based on the detection results of PonziGuard. Our findings suggest that even high-performance models may suffer from missed detections due to increasing syntactic diversity. Based on these insights, the following considerations are essential for future detector development.

First, the inability of models trained on 2015–2016 data to generalize to the 2017–2018 structures highlights the need for enhanced robustness and generalization through techniques such as transfer learning and continual learning. Moreover, the emergence of short-term syntactic shifts within specific clusters implies the possibility of adversarial implementations, underscoring the importance of carefully selecting syntactic features and employing multi-faceted evaluation methods.

Second, achieving more comprehensive detection requires the development of well-labeled datasets that minimize structural and temporal bias. Additionally, the use of clustering to monitor structural trends could serve as an effective auxiliary approach for the early detection of emerging fraudulent behaviors. In particular, it may help identify novel syntactic patterns that are prone to misclassification, enabling early warning systems to be implemented.

In summary, future research should prioritize the development of robust detection systems that not only improve detection accuracy and timeliness but also adapt effectively to the evolving syntactic strategies of smart Ponzi schemes.

8 Conclusion

This study evaluated the detection performance and adaptive limitations of PonziGuard, a state-of-the-art smart Ponzi scheme (SPS) detector, by analyzing the syntactic characteristics of contracts it identified. Through an investigation of structural evolution across time periods and complementary classification experiments using baseline models, we demonstrated that PonziGuard may fail to detect contracts with syntactically novel or complex patterns. Furthermore, we found a clear relationship between syntactic similarity and detection performance, highlighting the need for detector designs that account for ongoing structural evolution to maintain generalization performance.

These findings underscore the importance of addressing syntactic diversity and continuously updating detection models in future approaches to SPS detection. Our work contributes to a deeper understanding of the limitations of current detectors and provides actionable insights for enhancing the robustness and effectiveness of fraud detection on blockchain platforms.

Acknowledgment This work was supported by JSPS KAKENHI Grant Number JP25H01106.

References

1. XBlock: eXplore Blockchain Reliability — xblock.pro. http://xblock.pro/#/dataset/PonziContractDataset, accessed 17-10-2024
2. Bartoletti, M., Carta, S., Cimoli, T., Saia, R.: Dissecting ponzi schemes on ethereum: identification, analysis, and impact. Future Generation Computer Systems **102**, 259–277 (2020)
3. Benjamini, Y., Hochberg, Y.: Controlling the false discovery rate: a practical and powerful approach to multiple testing. Journal of the Royal statistical society: series B (Methodological) **57**(1), 289–300 (1995)
4. Chen, W., Zheng, Z., Ngai, E.C.H., Zheng, P., Zhou, Y.: Exploiting blockchain data to detect smart ponzi schemes on ethereum. IEEE Access **7**, 37575–37586 (2019)
5. etherscan.io: Ethereum (eth) blockchain explorer — etherscan.io. https://etherscan.io/, accessed 17-10-2024
6. Fan, S., Fu, S., Xu, H., Cheng, X.: Al-spsd: Anti-leakage smart ponzi schemes detection in blockchain. Information Processing and Management **58**(4), 1–13 (2021)
7. Fernández, A., Garcia, S., Herrera, F., Chawla, N.V.: Smote for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. Journal of artificial intelligence research **61**, 863–905 (2018)
8. Fisher, R.A.: On the interpretation of χ 2 from contingency tables, and the calculation of p. Journal of the royal statistical society **85**(1), 87–94 (1922)
9. Hu, T., Liu, X., Chen, T., Zhang, X., Huang, X., Niu, W., Lu, J., Zhou, K., Liu, Y.: Transaction-based classification and detection approach for ethereum smart contract. Information Processing and Management **58**(2), 1–19 (2021)
10. Liang, R., Chen, J., He, K., Wu, Y., Deng, G., Du, R., Wu, C.: Ponziguard: Detecting ponzi schemes on ethereum with contract runtime behavior graph (crbg). Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, pp. 1–12 (2024)
11. PonziDetection: Github - ponzidetection/ponziguard — github.com. https://github.com/PonziDetection/PonziGuard, accessed 17-10-2024
12. Wang, L., Cheng, H., Sun, Z., Tian, A., Yang, Z.: Pspl: A ponzi scheme smart contracts detection approach via compressed sensing oversampling-based peephole lstm. Future Generation Computer Systems **166**, 1–12 (2025)
13. Wang, L., Cheng, H., Zheng, Z., Yang, A., Zhu, X.: Ponzi scheme detection via oversampling-based long short-term memory for smart contracts. Knowledge-Based Systems **228**, 1–12 (2021)
14. Zheng, Z., Chen, W., Zhong, Z., Chen, Z., Lu, Y.: Securing the ethereum from smart ponzi schemes: Identification using static features. ACM Transactions on Software Engineering and Methodology **32**(5), 1–28 (2023)