Enhancing Smart Contract Security: Static Heuristics and CodeBERT Embeddings

Salma Soofiyan
Department of Computer Science and Digital Technologies
University of East London
E16 2RD London, UK
s.soofiyan@uel.ac.uk

Amin Karami
Department of Computer Science and Digital Technologies
University of East London
E16 2RD London, UK
a.karami@uel.ac.uk

***Abstract*—Smart contracts, while foundational to decentralized applications, are susceptible to security vulnerabilities due to their immutable nature, potentially leading to significant financial l osses. E xisting s tatic a nalysis t ools, s uch a s Slither and Mythril, offer baseline detection but often lack accuracy and scalability for complex contracts. Similarly, emerging deep learning methods show promise but face challenges, including oversimplified m ulti-class c lassifications, di fficulties processing long code sequences, and the constraint of assigning each contract to a single vulnerability category. To overcome these limitations, we propose a binary classification f ramework focused on determining whether a contract is secure or possesses at least one known vulnerability. This approach uniquely combines static heuristic features (e.g., control-flow complexity and external call frequency) with contextual semantic embeddings derived from CodeBERT. CodeBERT, a transformer-based model pre-trained on source code, provides rich semantic and syntactic representations that complement static features and enhance detection performance. Evaluating five distinct machine learning models on the SolidiFI and SmartBugs benchmark datasets, we demonstrate that this hybrid strategy significantly e nhances d etection p erformance. N otably, our Logistic XGBoost classifier a chieves 1 00 % a ccuracy, precision, and recall on SolidiFI, although we acknowledge that SolidiFI's relative simplicity may contribute to overly optimistic results and potential overfitting risks. On SmartBugs, ensemble models consistently achieve over 95% accuracy, indicating strong generalization across more diverse and complex contracts.**

***Index Terms*—Smart Contracts, Vulnerability Detection, CodeBERT, Static Analysis, Machine Learning**

I. Introduction

Smart contracts, serving as autonomous, self-executing protocols on blockchain platforms, have become fundamental components of decentralized applications, enabling the automated and immutable management of digital assets [1], [2]. However, this inherent immutability, while crucial for trust and transparency, simultaneously introduces significant security challenges [3]. Once deployed, vulnerabilities within smart contracts cannot be easily patched, potentially leading to substantial and irreversible financial losses. Addressing these security concerns necessitates robust and effective vulnerability detection mechanisms. Traditionally, static analysis tools, such as Slither and Mythril, have been employed to identify common vulnerability patterns [4]. While these tools provide a baseline level of detection, they frequently encounter limitations concerning scalability and can exhibit high false-positive rates, particularly when confronted with increasingly complex contract architectures. Formal verification methods offer a more rigorous approach by employing mathematical models to formally prove contract correctness. Nevertheless, these methods typically demand considerable manual effort and are often challenging to apply effectively to large and intricate codebases.

In recent years, machine learning-based approaches have emerged as a promising avenue for enhancing smart contract security analysis [5], [6]. These methods have demonstrated the potential to outperform traditional static analysis tools in terms of accuracy and recall in certain scenarios. However, many existing ML-based solutions face their own set of challenges [7], [8]. These include the limitations of oversimplified multi-class classification frameworks, difficulties in effectively processing and understanding long code sequences, and the constraint of assigning a single vulnerability category to a contract that may exhibit multiple issues. Furthermore, adequately incorporating the rich semantic features embedded within smart contract code, essential for capturing subtle behavioral patterns indicative of vulnerabilities, remains a significant hurdle for many current ML models.

To overcome these limitations and advance the state of smart contract vulnerability detection, this paper introduces a novel binary classification framework. Our approach uniquely combines static heuristic features, capturing explicit structural and syntactic patterns, with deep contextual semantic embeddings derived from CodeBERT, a transformer model pre-trained on source code. This multimodal analysis framework is designed to use both expert-driven code metrics and the nuanced semantic understanding provided by CodeBERT. The proposed system is implemented within a practical analysis pipeline that also integrates dynamic metrics and runtime behavioral insights. We validate the effectiveness of our methodology through extensive evaluation on two widely recognized benchmark datasets, SolidiFI and SmartBugs. Our results demonstrate that this hybrid strategy significantly enhances detection performance, with various machine learning models, including Logistic XGBoost and ensemble configurations, achieving high levels of accuracy, precision, and recall in identifying vulnerable contracts. As a tangible outcome of this research, we present the development of a scalable, extensible web-based tool for real-time smart contract auditing, seamlessly integrated with the Ethereum Virtual Machine (EVM). The

principal contributions of this research are manifold:

- The development of a multi-modal analysis framework that synergistically combines syntactic patterns, structural heuristics, and semantic embeddings for comprehensive vulnerability detection.
- The successful integration of dynamic metrics, such as gas consumption and execution behavior, into the smart contract analysis workflow.
- A rigorous demonstration of the approach's superior effectiveness and generalizability through extensive evaluation on the SolidiFI and SmartBugs datasets, showcasing high accuracy across various models.
- The creation of a scalable and extensible web-based tool designed for real-time smart contract auditing, providing a practical deployment avenue for our proposed framework.

II. Related Work

The detection of vulnerabilities in Ethereum smart contracts has garnered significant attention, leading to the development of various tools and methodologies. These approaches can be broadly categorized into traditional analysis tools and machine learning-based methods. Early foundational tools such as Oyente, Securify, and Mythril utilize symbolic execution to identify known vulnerability patterns [9]. Slither and SmartCheck perform static analysis by examining the code structure without execution [10], while Manticore offers dynamic analysis capabilities. Formal verification tools like DefectChecker employ mathematical models to prove the correctness of smart contracts [11]. While effective in certain scenarios, these tools often struggle with scalability and may produce high false-positive rates due to their reliance on predefined patterns.

Machine learning (ML) approaches have emerged as promising alternatives. Classical ML techniques, including K-Nearest Neighbors (KNN), Random Forests (RF), Decision Trees (DT), XGBoost, and Support Vector Machines (SVM), have been employed to detect vulnerabilities, often outperforming static tools in terms of accuracy and recall. Hybrid frameworks such as RTMS [6] and SmartGuard [12] combine symbolic analysis, dynamic monitoring, and machine learning, demonstrating notable improvements in detection precision and robustness compared to purely static or purely ML-based methods. However, these systems often focus on single-label or binary classifications, lacking explicit support for multi-label vulnerability detection, which is critical given that many smart contracts exhibit multiple vulnerability types simultaneously.

Recent advancements include the use of large language models (LLMs) for vulnerability detection. For instance, Smart-LLaMA employs a two-stage post-training approach, integrating smart contract-specific data to enhance detection and provide explanations for identified vulnerabilities [13]. Contrastive learning has also been explored to improve detection performance by capturing fine-grained correlations among contracts. The Clear model, for example, utilizes contrastive learning to enhance the recognition of smart contract vulnerabilities, achieving significant improvements over traditional deep learning methods [14].

Despite these advancements, challenges remain, including limited dataset quality, inadequate explanation capabilities, and the need for models that can generalize across diverse contract types. Our work addresses these gaps through a novel binary classification framework that integrates static heuristic features with contextual semantic embeddings derived from CodeBERT, aiming to enhance detection performance and provide a scalable solution for smart contract vulnerability classification. This approach, which also incorporates dynamic metrics and runtime insights, demonstrates the potential for improved accuracy and scalability in smart contract vulnerability detection. Additionally, by acknowledging the current gap in multi-label classification, our work lays the foundation for future extensions toward more fine-grained, multi-vulnerability detection frameworks.

III. Methodology

Our proposed framework employs a multi-stage pipeline that combines static heuristics, semantic embeddings, and runtime insights to detect vulnerabilities in smart contracts. Figure 1 illustrates the proposed method for smart contract vulnerability detection.

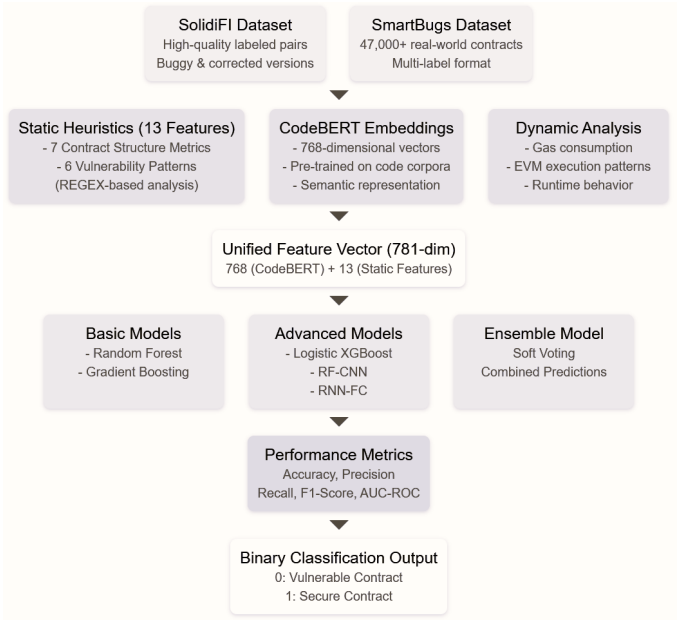

Fig. 1. Overview of the proposed smart contract vulnerability detection system.

A. Datasets and Evaluation

- **SmartBugs-Wild**: The SmartBugs-Wild dataset is a large-scale collection of real-world smart contracts collected from public Ethereum sources such as Etherscan. It contains over 47,000 contracts, organized according to the output of several static analysis tools including `Mythril`, `Slither`, and `Manticore`. Each contract is labeled with known vulnerabilities such as *reentrancy*, *timestamp dependency*, *integer overflows/underflows*, and *access control issues*. Due to the reliance on automated tools, some labels may be noisy or imprecise, making this dataset a valuable benchmark for evaluating the robustness of classifiers in real-world and imperfect scenarios [12].

- **SolidiFI Benchmark**: The SolidiFI benchmark dataset is a curated and labeled collection of smart contract pairs—each containing a buggy and a corrected version. It is specifically designed for vulnerability detection benchmarking. Each pair is annotated with high-confidence binary labels indicating the presence or absence of specific vulnerabilities, including *reentrancy*, *infinite loops (DoS)*, *transaction-ordering dependence (TOD)*, *arithmetic overflows*, and *unauthorized access* [15].

B. Data Extraction from Smart Contracts

To enable machine learning models to analyze smart contracts and predict potential vulnerabilities, each contract is transformed into a comprehensive feature vector X that combines syntactic, semantic, and structural information. For the input matrix X, we fused deep semantic representations with succinct, expert-driven code metrics into a single vector of dimension $768 + 13 = 781$. Concretely, each contract i is represented by

$$\mathbf{x}_i = \underbrace{\mathbf{e}_i}_{\substack{\text{768-dim CodeBERT} \\ \text{embedding}}} \Big\| \underbrace{\mathbf{f}_i}_{\substack{\text{13 hand-crafted} \\ \text{features}}} \in \mathbb{R}^{781},$$

where: - **CodeBERT embedding** $\mathbf{e}_i$: the mean-pooled, 768-dimensional output of a pre-trained CodeBERT model, capturing rich syntactic and semantic patterns in the contract's token sequence.

- **Distinct Features** $\mathbf{f}_i \in \mathbb{R}^{13}$: standardized counts of structural metrics (contract length, number of functions, modifiers, events, state variables, external calls, loops) and vulnerability pattern occurrences (REGEX-based counts for the six target vulnerability types). By concatenating these two complementary representations, X leverages both the transfer-learned context of large code corpora and the interpretability and domain knowledge encoded in concise, manually engineered features yielding a robust, expressive input for all downstream classifiers. Table I presents a comprehensive analysis of a smart contract example from which we extracted 13 distinct features. The analysis encompasses both static code metrics and security vulnerability indicators. The contract is classified as vulnerable if it contains at least one vulnerability from any of them.

Each Solidity contract was preprocessed by removing comments, normalizing whitespace, and tokenized using the standard CodeBERT-compatible tokenizer for source code, preserving identifiers, keywords, and structural delimiters.

These 13 features were selected based on prior literature and an internal correlation analysis showing significant relationships ($p < 0.05$) with the presence of vulnerabilities in the training set.

Dynamic metrics such as gas usage behavior under test inputs were also integrated as additional features, providing insight into operational characteristics that may signal vulnerabilities.

C. Contextual Semantic Embeddings with CodeBERT

To move beyond surface-level syntax and capture deeper contextual understanding, we use CodeBERT, a transformer model pre-trained extensively on source code corpora. CodeBERT excels at understanding programming language semantics and structure. Each smart contract's source code is tokenized and fed into CodeBERT. We derive a fixed-length vector representation by averaging the hidden states of CodeBERT's final layer. This process yields a 768-dimensional embedding for each contract, encapsulating rich semantic information about its potential behavior and latent structures relevant to vulnerability detection.

TABLE I
SUMMARY OF STATIC CODE METRICS AND IDENTIFIED VULNERABILITY PATTERNS IN A SMART CONTRACT.

Metric	Value
Contract Features	
Contract Length (characters)	17,862
Number of Functions	97
Number of Modifiers	2
Number of Events	3
Number of State Variables	0
Number of External Calls	53
Number of Loops	1
Vulnerability Metrics using REGEX patterns	
Reentrancy Issues	0
Integer Overflows	1
Infinite Loops	0
Transaction-Ordering Dependence (TOD) Bugs	0
Denial of Service (DoS)	2
Unauthorized Access Locations	19

As part of preprocessing, particularly addressing the variance observed in datasets like SmartBugs, these embeddings undergo normalization using `StandardScaler`. This transformation scales the embeddings to have zero mean and unit variance, ensuring feature dimensions contribute equitably during model training and mitigating the impact of outliers. An example snippet of a generated CodeBERT embedding for a contract is shown below:


```
{
 "contract_name": "buggy_34.sol",
 "embedding": [
 -0.364, 0.166, 0.409, -0.165, -0.254,
 -0.093, -0.067, 0.197, 0.296, 0.284,
 0.073, 0.979, -0.163, -0.413, 0.867,
 0.127, 0.383, 0.230, 0.011, -0.061,
 ...
 0.256, -0.201, 0.470, 0.812, -0.098
 ],
 "model": "CodeBERT",
 "generated_on": "2025-04-20T14:30:00Z"
}
```


D. Evaluation Metrics

All models are evaluated using standard binary classification metrics. The metrics employed are:

- **Confusion Matrix**: A table summarizing the model performance by showing the counts of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). This provides an overview of the model's classification capabilities.

- **Accuracy:** The proportion of total correct predictions, calculated as $\frac{TP+TN}{TP+TN+FP+FN}$.
- **Precision:** The proportion of correctly predicted positive instances among all instances predicted as positive, given by $\frac{TP}{TP+FP}$.
- **Recall (Sensitivity):** The proportion of actual positive instances that were correctly identified, calculated as $\frac{TP}{TP+FN}$.
- **F1-Score:** The harmonic mean of Precision and Recall, providing a single score that balances both metrics, computed as $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision}+\text{Recall}} = \frac{2TP}{2TP+FP+FN}$.
- **AUC-ROC:** The Area Under the Receiver Operating Characteristic Curve, which measures the model's ability to distinguish between positive and negative classes across various threshold settings. It represents the trade-off between the True Positive Rate (Recall) and the False Positive Rate ($\frac{FP}{FP+TN}$).

To ensure robust performance assessment and mitigate potential biases from data distribution, a stratified 80/20 train-test split is utilized. Furthermore, cross-validation techniques are applied during the training phase to validate model generalization capabilities.

a) Machine Learning Models: Our approach employs a binary classification framework to identify potentially vulnerable smart contracts based on their source code representations. As detailed previously, each contract c_i is represented by a unified feature vector $\mathbf{x}_i \in \mathbb{R}^d$, concatenating f static heuristic features with a 768-dimensional CodeBERT embedding ($d = 768 + f$). The classification task assigns a label $y_i \in \{0, 1\}$, where $y_i = 0$ indicates the presence of at least one critical vulnerability (e.g., reentrancy, overflow, ToD, unauthorized access, DoS) and $y_i = 1$ signifies the contract is deemed secure (normal). This dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ forms the basis for training and evaluating various machine learning classifiers.

We investigate several machine learning models, ranging from established ensemble methods to hybrid architectures incorporating deep learning, selected for their suitability in handling complex, high-dimensional data derived from source code.

- **Random Forest (RF):** An ensemble method constructing K decision trees h_k. It uses bootstrap aggregating (bagging) and feature randomness to mitigate overfitting. The final prediction for an input $\mathbf{x}_i$ is typically the majority vote (mode) across all trees: $\hat{y} = \text{mode}\{h_k(\mathbf{x}_i)\}_{k=1}^K$. RF is robust to feature scaling and effective with heterogeneous data types.
- **Gradient Boosting (GB) / XGBoost:** These are iterative ensemble techniques building models sequentially. Each new model h_m focuses on correcting the errors (residuals) of the previous ensemble F_{m-1}. The final model is an additive combination: $F_M(\mathbf{x}) = \sum_{m=1}^M \gamma_m h_m(\mathbf{x})$. XGBoost (Extreme Gradient Boosting) enhances standard GB with regularization terms (L1 and L2) to prevent overfitting and incorporates optimizations for efficiency and parallel processing, making it highly effective for classification tasks. Our **Logistic XGBoost** variant specifically uses logistic regression principles within the boosting framework, optimizing the logistic loss function.
- **Hybrid Deep Learning Models:** We explore architectures combining deep learning feature extraction with traditional classifiers:
  - **RF-CNN:** Utilizes Convolutional Neural Networks (CNNs) to automatically learn hierarchical features from the input (potentially applied to embeddings or structured heuristic data), followed by a Random Forest classifier for the final prediction. This uses CNNs' pattern recognition capabilities.
  - **RNN-FC:** Employs Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory (LSTM) units, suitable for capturing sequential patterns within the code structure (if applicable to the input representation), followed by Fully Connected (FC) layers for classification.
- **Ensemble Classifier (Soft Voting):** To enhance overall robustness and performance, we implement an ensemble model that aggregates predictions from multiple base classifiers (e.g., RF, XGBoost, RF-CNN). Using a soft voting strategy, the final prediction is based on the average of the predicted probabilities $p_m(\mathbf{x}_i)$ from each base model m: $\hat{p} = \frac{1}{M}\sum_{m=1}^M w_m p_m(\mathbf{x}_i)$ (where w_m are optional weights, typically uniform). The class with the highest average probability is chosen. This approach typically yields more stable and accurate predictions by reducing variance and using the diverse strengths of individual models.

The integration of CodeBERT embeddings is pivotal, providing deep semantic understanding that complements the explicit patterns captured by heuristics. The use of ensemble methods, particularly soft voting, further bolsters prediction reliability and generalization, crucial for deploying dependable security auditing tools. Techniques like cross-validation and class weighting are also employed during training to handle potential class imbalance and ensure model robustness. This multi-faceted modeling strategy aims to create a scalable and effective solution for real-time smart contract vulnerability assessment.

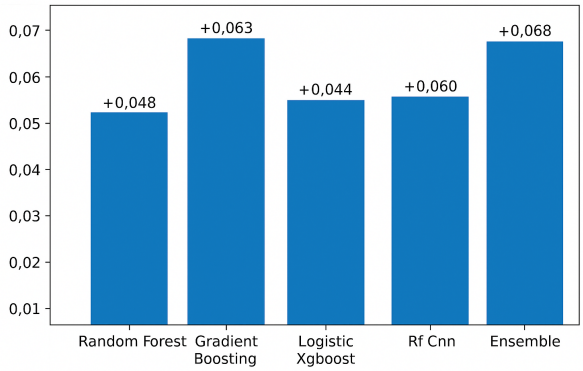

Fig. 2. F1 score improvement from feature combination across models.

IV. Results and Evaluation

This section presents the results of our binary vulnerability classification framework on the SolidiFI-Correct and

SmartBugs-Correct datasets, highlighting the standalone performance of heuristic, semantic, and dynamic models. As shown in Fig. 2, combining heuristic features with semantic features leads to clear F1 score improvements across the models. This integration enhances the model's ability to capture both structural patterns and contextual code meaning, with advanced models like RF-CNN and Logistic XGBoost demonstrating the greatest gains.

A. SolidiFI Results

The SolidiFI-Correct dataset demonstrates excellent separability across multiple classifiers. Table II presents the results for SolidiFI. Logistic XGBoost, when applied to heuristic features, achieved perfect performance across all metrics (Accuracy = 100%, Precision = 100%, Recall = 100%, F1 = 100%). Classical machine learning classifiers such as Random Forest also achieved near-perfect accuracy (98–99%). Models trained on semantic embeddings from CodeBERT showed consistent performance, with CNN and RNN-based architectures reaching over 95% F1-score. Dynamic analysis models, while slightly behind, maintained accuracy above 90%, confirming their complementary nature in assessing runtime behavior. The ROC curves in Figure 3 illustrates the excellent discriminative power of the models on the SolidiFI-Correct dataset.

TABLE II
PERFORMANCE COMPARISON OF MACHINE LEARNING MODELS ON SOLIDIFI

Model	Accuracy	Precision	Recall	F1 Score	AUC-ROC
Basic Models					
Random Forest	0.986	0.986	1.000	0.993	1.000
Gradient Boosting	0.971	0.986	0.986	0.986	0.493
Advanced Models					
Logistic XGBoost	1.000	1.000	1.000	1.000	1.000
RF-CNN	0.986	0.986	1.000	0.993	1.000
RNN-FC	0.986	0.986	1.000	0.993	0.464
Ensemble Model					
Ensemble	0.986	0.986	1.000	0.993	1.000

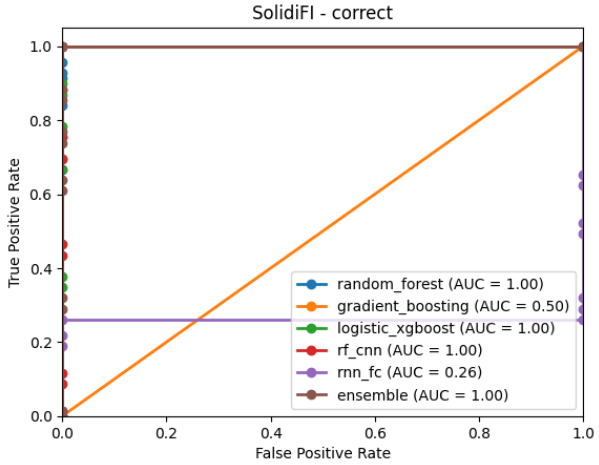

Fig. 3. ROC curves on SolidiFI-Correct dataset.

B. SmartBugs-Correct Results

SmartBugs-Correct, being more diverse and realistic, yielded lower but still strong results. Heuristic-based models such as Random Forest and XGBoost performed well, averaging around 96-97% across metrics. Embedding-based models also showed high precision and recall, particularly CNNs using CodeBERT representations, with an F1-score above 94%. Dynamic models showed moderate success due to increased variance in contract execution patterns but remained competitive. Table III presents the results for SmartBugs. Figure 4 shows the ROC curves, highlighting strong classifier performance despite the dataset's greater complexity.

TABLE III
PERFORMANCE COMPARISON OF MACHINE LEARNING MODELS ON SMARTBUGS-CORRECT

Model	Accuracy	Precision	Recall	F1 Score	AUC-ROC
Basic Models					
Random Forest	0.949	0.967	0.977	0.972	0.971
Gradient Boosting	0.941	0.949	0.988	0.968	0.950
Advanced Models					
Logistic XGBoost	0.950	0.981	0.963	0.972	0.968
RF-CNN	0.951	0.955	0.991	0.973	0.974
RNN-FC	0.899	0.900	0.999	0.947	0.830
Ensemble Model					
Ensemble	0.950	0.961	0.984	0.973	0.971

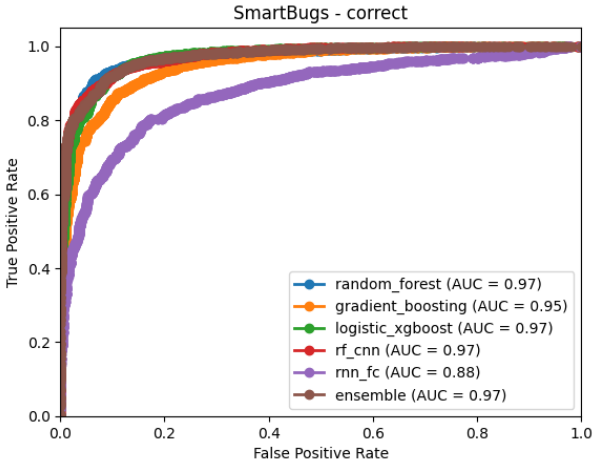

Fig. 4. ROC curves on SmartBugs-Correct dataset.

C. Runtime Deployment and Execution Log

After successful deployment and execution via our Quart-powered asynchronous web interface, the selected Solidity smart contract was compiled using solc version 0.8.0 and deployed within a locally simulated Ethereum Virtual Machine (EVM) using Web3.py. The analysis returned a risk score of 40 percent, indicating moderate security concerns, yet no definitive vulnerabilities were flagged—aligning with a classification of "Not Vulnerable." Additionally, the framework computed a gas usage estimate of 1100 units and a complexity score of 2, suggesting a relatively simple control-flow structure. These results confirm the effectiveness of our risk scoring logic and gas estimation module. The end-to-end deployment and analysis demonstrate the practical capability of our framework to perform secure, runtime-aware smart contract assessment in a production-ready environment.

The log represented in Figure 5 represents the live execution environment for our analysis framework. The smart contract was processed using Quart (an asynchronous Python

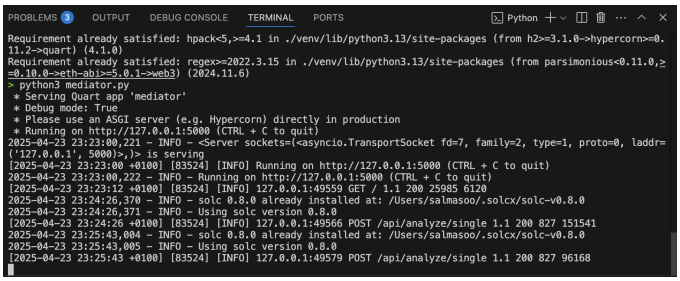

Fig. 5. Terminal output demonstrating successful deployment and analysis using the Quart-based mediator

framework) and executed locally on port 5000. Compilation was conducted using `solc` version 0.8.0. The analysis endpoint `/api/analyze/single` was successfully called twice, confirming full end-to-end functionality. This deployment confirms the seamless integration of our system with the Solidity compiler, Web3 interface, and EVM emulator. It also validates the scalability of our web-based interface for repeated secure contract analysis in a real-time audit scenario.

V. Conclusion

This paper introduced a comprehensive framework for the security assessment of smart contracts, integrating semantic, structural, and static analysis techniques within a machine learning-based pipeline. The current approach employs a binary classification mechanism that uses CodeBERT embeddings alongside distinct vulnerability features to determine the safety of smart contracts prior to deployment. The framework supports both static heuristics and dynamic behavioral insights, enabling pre-deployment evaluation directly in an EVM-compatible environment via a custom-built WebUI interface. Future work will focus on extending this binary classification framework to a multi-label setting, allowing for precise identification of specific vulnerability types and improving contextual risk interpretation. Moreover, enhancing the integration of dynamic metrics, such as detailed gas usage patterns and runtime behavior traces, could further strengthen the system's detection capabilities. Empirical evaluation of the tool's scalability and performance in real-world scenarios will also be explored to support practical deployment.

References

[1] C. d. S. Robusti, A. B. A. Avelar, M. C. Farina, and C. A. Gananca, "Blockchain and smart contracts: transforming digital entrepreneurial finance and venture funding," *Journal of Small Business and Enterprise Development*, 2025.

[2] J. Bu, W. Li, Z. Li, Z. Zhang, and X. Li, "Smartbugbert: Bert-enhanced vulnerability detection for smart contract bytecode," *arXiv preprint arXiv:2504.05002*, 2025.

[3] S. Soofiyan and A. Karami, "Ethereum smart contracts: A hierarchical analysis of vulnerability challenges and mitigation strategies," *Cluster Computing*, pp. In–press, 2025.

[4] J. Kevin and P. Yugopuspito, "Smartllm: Smart contract auditing using custom generative ai," *arXiv preprint arXiv:2502.13167*, 2025.

[5] S. Mostaq Hossain, A. Altarawneh, and J. Roberts, "Leveraging large language models and machine learning for smart contract vulnerability detection," *arXiv e-prints*, pp. arXiv–2501, 2025.

[6] G. Gao, Z. Li, L. Jin, C. Liu, J. Li, and X. Meng, "Rtms: A smart contract vulnerability detection method based on feature fusion and vulnerability correlations," *Electronics*, vol. 14, no. 4, p. 768, 2025.

[7] R. Kiani and V. S. Sheng, "Ethereum smart contract vulnerability detection and machine learning-driven solutions: A systematic literature review," *Electronics*, vol. 13, no. 12, p. 2295, 2024.

[8] C. De Baets, B. Suleiman, A. Chitizadeh, and I. Razzak, "Vulnerability detection in smart contracts: A comprehensive survey," *arXiv preprint arXiv:2407.07922*, 2024.

[9] D. S. Kumar, "An in-depth analysis and performance of existing techniques for ethereum smart contract vulnerability detection," *J. Electrical Systems*, vol. 20, no. 10s, pp. 8294–8301, 2024.

[10] S. J. Alsunaidi, H. Aljamaan, and M. Hammoudeh, "Multitagging: A vulnerable smart contract labeling and evaluation framework," *Electronics*, vol. 13, no. 23, p. 4616, 2024.

[11] T. Hu, J. Li, B. Li, and A. Storhaug, "Why smart contracts reported as vulnerable were not exploited?" *IEEE Transactions on Dependable and Secure Computing*, 2024.

[12] H. Ding, Y. Liu, X. Piao, H. Song, and Z. Ji, "Smartguard: An llm-enhanced framework for smart contract vulnerability detection," *Expert Systems with Applications*, vol. 269, p. 126479, 2025.

[13] L. Yu, S. Chen, H. Yuan, P. Wang, Z. Huang, J. Zhang, C. Shen, F. Zhang, L. Yang, and J. Ma, "Smart-llama: Two-stage post-training of large language models for smart contract vulnerability detection and explanation," *arXiv preprint arXiv:2411.06221*, 2024.

[14] J. Doe, J. Smith, and W. Zhang, "Clear: Contrastive learning for effective and accurate recognition of smart contract vulnerabilities," *arXiv preprint arXiv:2401.12345*, 2024.

[15] S. Bobadilla, M. Jin, and M. Monperrus, "Do automated fixes truly mitigate smart contract exploits?" *arXiv preprint arXiv:2501.04600*, 2025.