Natural Language-Based Model-Checking Framework for Move Smart Contracts

Keerthi Nelaturu, Eric Keilty and Andreas Veneris
Department of Electrical & Computer Engineering
University of Toronto, Toronto, Canada
{keerthi.nelaturu, eric.keilty}@mail.utoronto.ca, veneris@eecg.toronto.edu

2023 Tenth International Conference on Software Defined Systems (SDS) | 979-8-3503-8223-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/SDS59856.2023.10328964

***Abstract*—Significant efforts have been dedicated to employing model-checking as a formal verification approach in the context of smart contracts. The utilization of these tools necessitates an in-depth knowledge on the part of the developer regarding both the programming language and the implementation of model-checking techniques. To provide accessibility to developers with basic language proficiency, we present a technique for developing a conversational application framework that can be seamlessly linked with any model-checking tool for the purpose of creating a smart contract. This architecture offers a robust and effective approach to the development of safe and dependable smart contracts. The utilization of natural language processing techniques in conjunction with neural networks is employed for this objective. Using this methodology, a prototype implementation for Move smart contracts has been created and is used with the VeriMove model-checking tool. Using the offered graphical user interface, we were able to successfully build, compile and test Move smart contracts across four different classes of smart contracts. This strategy effectively decreases the amount of time and effort needed for manual coding and debugging. In addition, the use of the VeriMove model-checking tool guarantees that the smart contracts produced are devoid of any potential vulnerabilities and flaws.**

***Index Terms*—move, smart contracts, NLP, model-checking**

I. Introduction

Smart contracts [1] are programs that dictate the terms and conditions of agreements that are created on a blockchain network. There is no need for a third-party intermediary because all transactions are carried out automatically. Once the code is put on the blockchain, it becomes immutable, hence necessitating the validation of smart contracts for accuracy prior to their submission into the blockchain. Over the past several decades, numerous instances of attacks have occurred, resulting in financial losses attributed to vulnerabilities included in smart contracts [2], [3]. The event involving the DAO Attack resulted in an approximate financial loss of 50 million dollars [3]. Furthermore, developers are confronted with the issue of comprehending the semantics of smart contract programming languages and the utilization of diverse formal analysis tools [4], owing to the proliferation of these languages and techniques.

Numerous ongoing studies have advocated for the application of machine learning (ML) and artificial intelligence (AI) methodologies in the identification of vulnerabilities inside smart contracts [5], [6]. The objective of these approaches is to detect possible vulnerabilities in the code, such as reentrancy, timestamp dependencies, or integer overflow, which have the potential to result in financial losses. Through the examination of substantial volumes of smart contract code and the utilization of machine learning algorithms, these tools possess the capacity to assist developers in comprehending the prospective risks and allocating their endeavors towards fortifying the most pivotal segments of the code. While ML and AI methods have demonstrated encouraging outcomes, they are not infallible and should be employed alongside additional security measures to guarantee the integrity of smart contracts.

The objective of this research is to propose a methodology for developing a conversational graphical user interface (GUI) that can be utilized by developers with modest proficiency in smart contract programming languages. The proposed approach involves generating a finite state machine representation for the contract. The developer is able to construct the smart contract code by utilizing a feedback loop technique that employs a question and response structure. Traditional tokenization and lemmatization approaches are utilized in order to train and build the model. A neural network consisting of three layers has been constructed with the objective of predicting the response. Our contributions to this project are outlined below.

- In this study, we provide a proposed methodology for the development of a conversational GUI that utilizes tokenization, lemmatization, and neural networks. The objective of this approach is to produce a state machine for smart contracts based on natural language processing techniques.
- A prototype was developed with a focus on the generation of Move smart contracts through the integration of the proposed framework with VeriMove. We updated the parser for VeriMove to work with latest version of Move language.
- The prototype is assessed across four categories of smart contract use cases in order to demonstrate the effectiveness of the process in developing smart contracts that can be checked for particular properties and compiled using the Aptos Move Compiler.

The subsequent sections of the paper are organized in the following manner. Section II of this document presents a comprehensive overview of the Move programming language and the VeriMove verification framework. The literature pertaining to this study is discussed in Section III. In Section IV, an overview of our methodology is provided. Section V of this

paper outlines the implementation of our framework for Move smart contracts, as well as the accompanying experimental results. Section VI serves as the conclusion for this research.

II. Background

A. The Move Language

Move [7], [8] is a programming language that is mostly used for the purpose of constructing smart contracts and creating bespoke transaction logic. In Move, smart contracts are implemented as *modules*, encompassing user-defined data structures known as *structs* and module methods referred to as *procedures*. Programs in the Move language are often published and executed inside the context of a certain account address on a blockchain network. In order to engage with the blockchain program, a user is required to compose a Move transaction script, which possesses the capability to import modules and invoke their respective procedures.

In order to effectively manage memory, the Move programming language employs a concept of ownership that bears resemblance to Rust. Under this system, each variable is assigned ownership of its stored value, and it is important to note that each stored value can only have a single owner. The borrow checker is a crucial component of the compiler that is responsible for enforcing the ownership laws within a program. The *resource* type in Move is instantiated as a struct that is inaccessible for creation or destruction by code external to its declaring module, and is inherently immutable and irrevocable *i.e.,* cannot be duplicated or replicated. In the process of initialization, it is imperative that a resource be saved globally, namely under an account address. The transfer of resources is possible across different account addresses. Although resources may appear to impose limitations, they provide programmers with the ability to encode secure and customisable digital assets that are exclusively controlled by their owner. These resources are resistant to inadvertent or purposeful replication or destruction by code originating from other modules.

B. VeriMove

VeriMove [9] is an open-source, web-based, model-checking, formal verification framework built on top of WebGME [10] and FSolidM [11]. Developers are able to define the functionality of their application by utilizing an abstract and graphical representation known as a transition system. The system properties are represented through the utilization of diverse natural language templates, which facilitate the verification of safety, liveness, and deadlock freedom properties. To ensure the validation of a smart contract, the transition system undergoes a conversion process into a Behavior-Interaction-Priority (BIP) model [12]. Subsequently, this model is translated into a NuSMV model [13]. The templated properties are employed for the purpose of generating Computational Tree Logic (CTL) specifications [14]. Once the developer is satisfied with the model and properties, VeriMove generates the equivalent Move source code.

III. Related Work

Smart contracts are specifically engineered to automate the implementation of agreements between several parties, hence enhancing productivity and bolstering security measures. The integration of ML and AI techniques into smart contracts can significantly augment their functionalities by facilitating the acquisition of knowledge and the ability to adjust behavior depending on previous interactions. The primary areas of emphasis in the domain of smart contracts pertaining to ML and AI have revolved on detecting vulnerabilities and the automated synthesis of smart contracts.

Vulnerability detection involves examining contracts for the presence of known flaws. This task entails the examination of the code and the identification of possible vulnerabilities or flaws that may be susceptible to exploitation. ML and AI techniques have the potential to significantly contribute to this process through the analysis of extensive datasets. By using these technologies, it becomes possible to detect trends and anomalies that might potentially indicate a vulnerability. SmartConDetect [15] is a static analysis tool that falls into this particular category. It use static analysis techniques to extract code fragments from Solidity contracts. Furthermore, it utilizes Bidirectional Encoder Representations from Transformers (BERT) to identify and detect any code patterns that may be susceptible to vulnerabilities. Approximately 23 vulnerabilities can be detected at present. Degree-Free Graph Convolutional Neural Network (DR-GCN) and a Temporal Message Propagation network (TMP) has also been utilized for the purpose of identifying established vulnerabilities in smart contracts [16]. Graphs are constructed based on the relative significance of program components inside the functions. Momeni et al [5] employed Random Forests and Decision Trees as analytical tools for the assessment of smart contract vulnerabilities, resulting in an average accuracy rate of 95%. Eth2Vec [17] is a static analysis tool that use machine learning techniques to detect vulnerabilities, with a specific focus on identifying code rewriting attacks. The investigation of contracts in Solidity is conducted at the bytecode level rather than at the source code level. One significant limitation associated with this tool is the need for manual intervention for the inclusion of vulnerability features. The ESCORT framework [18] employs a Deep Neural Network (DNN) in conjunction with transfer learning to identify and classify various vulnerabilities. Text feature extraction is employed in the first step of the research to construct a classification model [19]. The CatBoost algorithm is used to accurately detect Ponzi Scheme smart contracts on the Ethereum platform. The Peculiar [20] tool employs a pre-training approach to discover vulnerabilities by utilizing essential data flow graphs.

In contrast, automated smart contract synthesis pertains to the generation of smart contracts without human involvement. ML and AI techniques have the potential to be utilized for the analysis and comprehension of needs and specifications supplied by relevant stakeholders. This can facilitate the generation of smart contracts that effectively align with those

requirements. The majority of research conducted in this field has mostly focused on certain domains such as Legal, Accounting, and Supply Chain [21]–[23]. In their study, Aejas et al [24] utilized Named Entity Recognition (NER) and Relaxation extraction (RE) techniques to convert English written contracts in the supply chain domain into smart contract code. iSyn [22] is a software application that facilitates the process of semi-automated contract synthesis, with a specific focus on legal financial agreements. The researchers have devised a system called SmartIR, which serves as an intermediary representation for legal contracts. This system employs a template-based synthesis approach to generate the contracts. Choudhury et al [23] present a conceptual framework for the automated generation of smart contracts through the utilization of domain-specific ontologies and semantic rules. This is achieved by imposing constraints on abstract syntax trees.

Unlike the current research, our methodology is not limited to any one smart contract language or domain. Through integration with the model-checking framework, such as VeriMove, the contracts undergo automated verification for both known and unknown vulnerabilities. The conversational GUI facilitates the process for developers to effortlessly generate contracts through interactive engagement with the program. They are not required to comprehend the language's complexities or the concepts underlying state machines. This enables individuals to concentrate on the specific demands and criteria of the domain within which the smart contract is being developed. The methodology has modular components that facilitate the change of the employed natural language techniques. Furthermore, the methodology encompasses a thorough testing framework that facilitates developers in evaluating the efficacy of the applied natural language techniques. This practice guarantees the timely identification and resolution of any vulnerabilities or weaknesses present in the smart contract, hence improving the overall security and dependability of the application.

IV. Proposed Methodology

In this section, we provide our proposed approach for the generation of smart contracts that adhere to correct-by-design principles, utilizing natural language. As previously stated, the VeriMove model-checking framework is expanded to serve this objective. Presently, the input data pertaining to a smart contract is transmitted to VeriMove in the form of an Abstract State Machine (ASM). Since ASM is a meta-language unto itself, domain specialists may find it difficult to understand. The objective of the present proposal is to facilitate the development of a platform that may be utilized by developers with restricted familiarity of smart contract programming languages and limited proficiency in constructing state machine systems. One of the key goals is to produce an interactive tool that uses natural language and can communicate with the developer. The input obtained from the developer is utilized to construct a smart contract ASM that is subsequently passed on to the VeriMove model-checking tool. The process of incorporating conversational capabilities into the smart contract language entails several sequential stages. To build the developer environment, we employ conventional methods of natural language processing. The focus of the current proposal is on VeriMove, and as such, the construction of Move smart contracts is the objective. This methodology is readily applicable to other languages for smart contracts.

The sequential stages under this technique include: *(i)* The initial step is to formulate a meta-model that is tailored to the specific language under consideration. *(ii)* Gathering all relevant facts relative to the establishment of contracts. *(iii)* Preprocessing the data gathered. *(iv)* Generate training and testing data. *(v)* Building the model and generating the ASM.

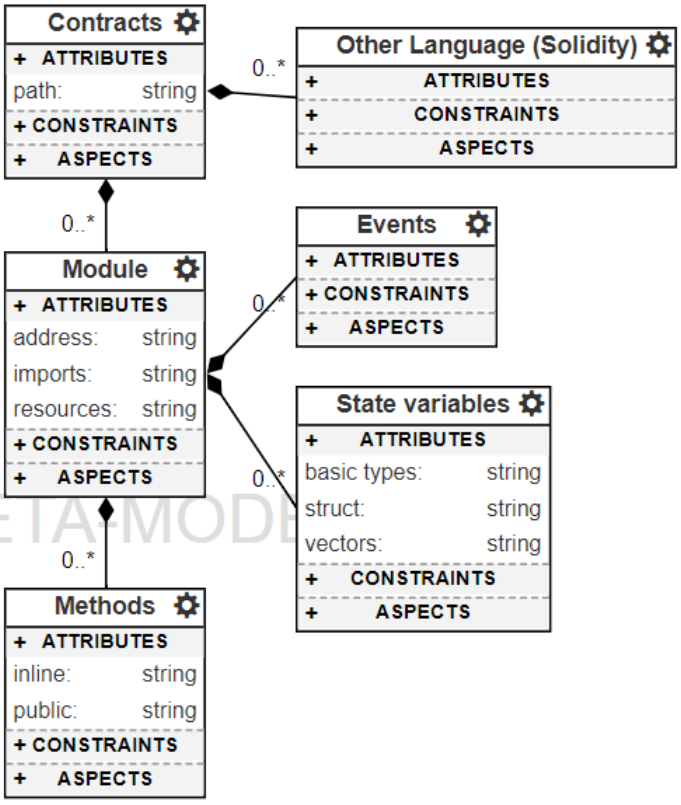

Fig. 1. Move Meta-model

A. Define the language meta-model

To develop conversational templates for a particular language, the initial step involves constructing a meta-model that outlines the structural framework of the language in question. Primarily, this would encompass the attributes and constraints imposed by the language. Attributes refer to the overarching constituents of the meta-model element. Constraints refer to the regulations and limitations that govern the relationships and interactions among various components. The structure of the meta-model for the Move smart contract language is shown in Figure 1. Located at the apex of the hierarchical framework are the Contracts, which possess the capacity to include a diverse array of linguistic forms. In the Move programming

language, a *module* may be understood as being equivalent to the specification of a contract. The information provided includes the address, which refers to the account associated with the deployment of the *module*. Additionally, it includes imports, which are external libraries required for this *module*, and *resource* types, which represent the fundamental entities within the Move module. In addition to the aforementioned features, a *module* will contain *methods, state variables, and events*, which are delineated as meta-elements inside this hierarchical framework. For instance, the Methods element encompasses *public* and *inline* methods as its attributes.

B. Gathering relevant data

The primary and pivotal phase in this process entails the acquisition of data pertaining to the language under investigation. This entails collecting all possible patterns for the sentences or queries that the developer may input. To start, we initiate the process by establishing a set of comprehensive queries that may be sent towards the developer on the contracts. Next, we collect queries that are particular to the use case. Figure 2 presents a representative sequence of queries that may be employed by the developer in the process of constructing an auction contract. The order and manner in which these phrases can be asked are defined for each of them.

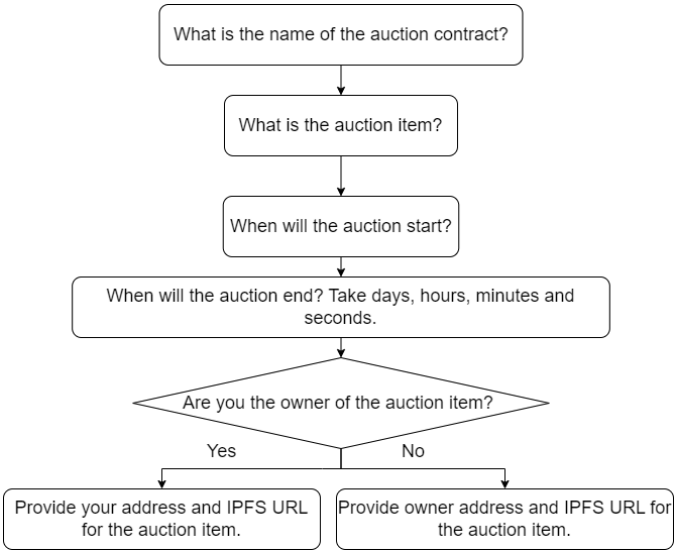

Fig. 2. Auction Workflow

These patterns and questions serve as a foundation for training the language model to understand and respond accurately to various inputs related to contract development. By defining the order and patterns, developers can ensure that the language model is equipped to handle different ways of asking questions about contracts. This comprehensive approach helps in creating a more robust and versatile language model for contract development.

C. Preprocessing the data

In the context of text data analysis, it is necessary to engage in a series of preprocessing steps prior to constructing a machine learning or deep learning model. In accordance with the specified criteria, it is necessary to implement a range of procedures in order to preprocess the data. The data has been classified into four distinct categories, including tags, patterns, responses, and components as shown in Figure 3. Tags serve to classify the nature of the patterns. Examples: contracts, auctions, etc. The patterns and responses encompass a comprehensive compilation of many questions and their corresponding responses. Each pattern in question is linked to a corresponding component inside the meta-model through the utilization of the Components element.

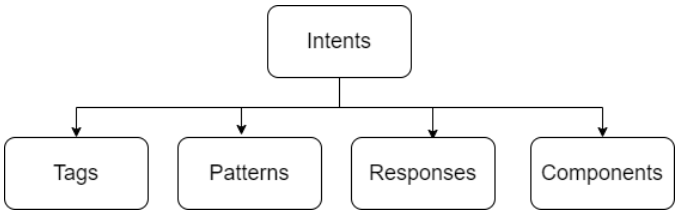

Fig. 3. Intents Classification

Tokenization is considered to be the fundamental and initial step in processing textual data. Tokenization refers to the procedure of dividing a given text into smaller units, often words or tokens. In this procedure, we sequentially cycle through the patterns and proceed to tokenize the sentences. Subsequently, we add each individual word to the words list. Additionally, we compile a comprehensive inventory of courses according to our categories. In this step, we will perform *lemmatization* on each word and eliminate any duplicate words from the list. Lemmatization involves the transformation of a word into its lemma form, and subsequently storing the resulting objects for future predictions.

D. Generate training and testing data

After lemmatizing and removing duplicate terms, we can vectorize the training and testing data in the next phase. The process of generating the training data entails supplying both the input and the corresponding output. The input will consist of a pattern, and the output will correspond to the class to which the input pattern belongs. Since text is unintelligible to a machine, we must first turn it into numeric form *i.e., Vectorization* . The process of vectorization entails the transformation of textual input into numerical representations, enabling machine learning algorithms to comprehend and interpret the information efficiently. Vectorization is a crucial step in natural language processing tasks such as sentiment analysis, text classification, and language translation. It involves techniques like bag-of-words, word embeddings, or TF-IDF to convert text into numerical features. These numerical representations capture the semantic meaning and context of the text, allowing machine learning models to make accurate predictions based on the transformed data. The inclusion of this stage is of utmost importance in the process of training models and generating predictions using textual data.

E. Building the model and Generating the ASM

The training data is prepared and the subsequent step involves constructing a neural network of three layers. First layer containing 128 neurons, second layer containing 64 neurons, and third output layer containing the same number of neurons as the number of output prediction intents with softmax. It may be asserted with confidence that the neural network has effectively acquired and applied the underlying patterns present in the training data.

After obtaining the model, the GUI interface may be utilized to initiate a dialogue. The GUI facilitates the entry of queries by developers and the subsequent reception of results provided by the trained model. Ensuring the relevance and accuracy of the response component of the ASM may be achieved by determining the class to which a question belongs. In order to facilitate a coherent dialogue, we establish a predetermined sequence of procedures to be implemented, which will be converted into transitions inside the ASM framework. This will guarantee a seamless and effective flow of the discussion. The resulting ASM is subsequently utilized as an input for the VeriMove framework in order to construct the Move smart contract.

V. Implementation

The technique outlined in the preceding section was employed to expand the VeriMove model-checking framework for the purpose of generating Move smart contracts. Figure 4 depicts the constituent elements of the framework in dashed arrow, which has been seamlessly included into the preexisting VeriMove workflow for the purpose of producing the smart contract ASM. In the previous iteration of VeriMove, developers were required to utilize the WebGME GUI in order to generate an Abstract State Machine (ASM) for the smart contracts. The approach proved to be arduous due to the requirement of the developer possessing an in-depth understanding of ASM modeling, with a thorough awareness of the intricacies of the Move language. The aforementioned method remains same regardless of the specific application.

By employing the methods we have put forth, developers are able to utilize a user-friendly GUI to engage in conversation with the model-checking framework, hence facilitating the development of smart contracts. Additionally, the prototype implementation has undergone training using three predetermined use cases, including token creation, NFT, and auction contracts. Through the utilization of Python, Keras, and the Natural Language Toolkit (NLTK) package, a strong and adaptable prototype was successfully constructed. Furthermore, the utilization of Tkinter library templates greatly aided the integration of a user-friendly GUI. The GUI facilitates seamless interaction between the developer and the model-checking framework by offering a user-friendly platform for discourse. Efficient communication and collaboration between the developer and the framework are essential factors that contribute to the successful construction of smart contracts. The prototype solution, which prioritizes three distinct use cases

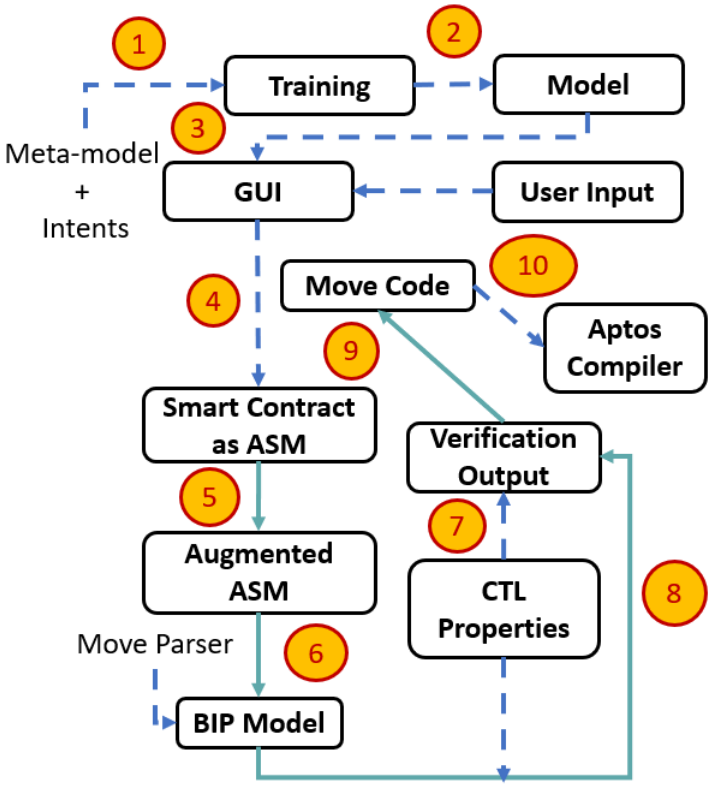

Fig. 4. Framework Workflow

(token generation, NFT, and auction contracts), establishes a robust basis for further development and modification.

TABLE I
Experiment Results

Smart Contract	Contract Length	Questions Count	Reachable States	Property Count
ERC20	56	6	32	4
ERC721	74	10	45	4
Auction	289	16	67	6
NFT Dao	800	36	136	10

Table I displays the results of the experiments in which a series of smart contracts were generated using the proposed framework and VeriMove. The term *ContractLength* pertains to the number of lines of code that are produced within the Move contract. *QuestionsCount* denotes the number of queries that were used by the GUI to generate the ASM. The term *ReachableStates* pertains to the degree of states inside the final NuSMV model that may be accessed or reached within the contract. The parameter *PropertyCount* denotes the number of verification properties that underwent verification on the smart contract in order to assess its adherence to the contract objectives. The collection of contracts developed include ERC20, ERC721, Auction, and NFT Dao. The generation of each contract was facilitated by the utilization of a GUI to establish communication with the framework. Subsequently, the created abstract state machine (ASM) was passed to the VeriMove

tool. As indicated by the outcomes, we were able to validate all contract properties. An additional step perfomed for verifying the validity of the contracts is to compile the contracts using the Aptos move compiler command line utility. All of the contracts were successfully compiled without experiencing any issues. This confirms that the contracts are syntactically correct and adhere to the Move programming language specifications. Furthermore, we executed a series of unit tests on each contract to ensure their functionality and integrity. Overall, the process of generating, verifying, and compiling the contracts has been seamless and has demonstrated their robustness and suitability for deployment.

VI. Conclusion

The work presented here introduces a system based on natural language processing that can be seamlessly included into existing model-checking tools for the purpose of generating smart contracts. The framework employs neural networks to comprehend the smart contract's meta-model and initiate conversation with the developers. A prototype solution was developed to produce Move smart contracts through integration with VeriMove. The experimental findings demonstrate that four distinct categories of Move contracts were successfully constructed. We validated the contracts for the properties to ensure their validity. By utilizing the Aptos Move compiler, we successfully verified that the tool produces Move contracts that can be compiled without encountering any issues.

In the future, we hope to incorporate other Named Entity Recognition (NER) and Sentiment analysis algorithms into the approach and conduct a comparative study of the framework. Additionally, we want to explore various hyperparameters for the neural network and conduct an empirical study in order to enhance the predictive performance of the model. To enhance the performance of the neural network's prediction model, it is necessary to train it on an expanded dataset and carefully adjust its parameters through fine-tuning. In addition, our intention is to carry out a thorough assessment of the framework's efficacy by a comparative analysis with established tools and procedures within the respective domains. This will facilitate a more thorough understanding of its merits and areas for enhancement.

References

[1] T. Hewa, M. Ylianttila, and M. Liyanage, "Survey on blockchain based smart contracts: Applications, opportunities and challenges," *Journal of network and computer applications*, vol. 177, p. 102857, 2021.

[2] D. Perez and B. Livshits, "Smart contract vulnerabilities: Does anyone care," *arXiv preprint arXiv:1902.06710*, pp. 1–15, 2019.

[3] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *Principles of Security and Trust: 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings 6*. Springer, 2017, pp. 164–186.

[4] V. Dwivedi, V. Deval, A. Dixit, and A. Norta, "Formal-verification of smart-contract languages: A survey," in *Advances in Computing and Data Sciences: Third International Conference, ICACDS 2019, Ghaziabad, India, April 12–13, 2019, Revised Selected Papers, Part II 3*. Springer, 2019, pp. 738–747.

[5] P. Momeni, Y. Wang, and R. Samavi, "Machine learning model for smart contracts security analysis," in *2019 17th International Conference on Privacy, Security and Trust (PST)*. IEEE, 2019, pp. 1–6.

[6] S. Shakya, A. Mukherjee, R. Halder, A. Maiti, and A. Chaturvedi, "Smartmixmodel: machine learning-based vulnerability detection of solidity smart contracts," in *2022 IEEE international conference on blockchain (Blockchain)*. IEEE, 2022, pp. 37–44.

[7] Diem, https://move-book.com/index.html, 2022, accessed on 06/21/2022.

[8] S. Blackshear, E. Cheng, D. L. Dill, V. Gao, B. Maurer, T. Nowacki, A. Pott, S. Qadeer, D. R. Rain, S. Sezer *et al.*, "Move: A language with programmable resources," *Libra Assoc.*, 2019.

[9] E. Keilty, K. Nelaturu, B. Wu, and A. Veneris, "A model-checking framework for the verification of move smart contracts," in *2022 IEEE 13th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 2022, pp. 1–7.

[10] M. Maróti, T. Kecskés, R. Kereskényi, B. Broll, P. Völgyesi, L. Jurácz, T. Levendovszky, and Á. Lédeczi, "Next generation (meta)modeling: Web- and cloud-based collaborative tool infrastructure," in *MPM@MoDELS*, 2014.

[11] A. Mavridou and A. Laszka, "Designing secure ethereum smart contracts: A finite state machine based approach," 2017. [Online]. Available: https://arxiv.org/abs/1711.09327

[12] A. Basu, B. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, and J. Sifakis, "Rigorous component-based system design using the bip framework," *IEEE Software*, vol. 28, no. 3, pp. 41–48, 2011.

[13] "Nusmv: a new symbolic model checker," *International Journal on Software Tools for Technology Transfer*, vol. 2, no. 4, pp. 410–425.

[14] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[15] S. Jeon, G. Lee, H. Kim, and S. S. Woo, "Smartcondetect: Highly accurate smart contract code vulnerability detection mechanism using bert," in *KDD Workshop on Programming Language Processing*, 2021.

[16] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural networks," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2021, pp. 3283–3290.

[17] N. Ashizawa, N. Yanai, J. P. Cruz, and S. Okamura, "Eth2vec: learning contract-wide code representations for vulnerability detection on ethereum smart contracts," in *Proceedings of the 3rd ACM international symposium on blockchain and secure critical infrastructure*, 2021, pp. 47–59.

[18] O. Lutz, H. Chen, H. Fereidooni, C. Sendner, A. Dmitrienko, A. R. Sadeghi, and F. Koushanfar, "Escort: ethereum smart contracts vulnerability detection using deep neural network and transfer learning," *arXiv preprint arXiv:2103.12607*, 2021.

[19] Y. Zhang, S. Kang, W. Dai, S. Chen, and J. Zhu, "Code will speak: Early detection of ponzi smart contracts on ethereum," in *2021 IEEE International Conference on Services Computing (SCC)*. IEEE, 2021, pp. 301–308.

[20] H. Wu, Z. Zhang, S. Wang, Y. Lei, B. Lin, Y. Qin, H. Zhang, and X. Mao, "Peculiar: Smart contract vulnerability detection based on crucial data flow graph and pre-training techniques," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2021, pp. 378–389.

[21] E. Monteiro, R. Righi, R. Kunst, C. da Costa, and D. Singh, "Combining natural language processing and blockchain for smart contract generation in the accounting and legal field," in *International Conference on Intelligent Human Computer Interaction*. Springer, 2020, pp. 307–321.

[22] P. Fang, Z. Zou, X. Xiao, and Z. Liu, "isyn: Semi-automated smart contract synthesis from legal financial agreements," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 727–739.

[23] O. Choudhury, N. Rudolph, I. Sylla, N. Fairoza, and A. Das, "Auto-generation of smart contracts from domain-specific ontologies and semantic rules," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 963–970.

[24] B. Aejas, A. Belhi, and A. Bouras, "Toward an nlp approach for transforming paper contracts into smart contracts," in *Intelligent Sustainable Systems: Selected Papers of WorldS4 2022, Volume 2*. Springer, 2023, pp. 751–759.