Blockchain-Enabled Decentralized Trustworthy Framework Envisioned for Patient-Centric Community Healthcare

Mohammad Khalid Imam Rahmani[1]*, Javed Ali[2], Surbhi Bhatia Khan[3], Muhammad Tahir[4]
College of Computing and Informatics, Saudi Electronic University, Riyadh 11673, Saudi Arabia[1, 2, 4]
Department of Information Systems, College of Computer Science and Information Technology,
King Faisal University, Saudi Arabia[3]
School of Science, Engineering and Environment, University of Salford, United Kingdom[3]
Department of Computer Science, National University of Science & Technology (NUST), Balochistan Campus, Quetta, Pakistan[4]

***Abstract*—Ethereum has gained significant attention from businesses as a blockchain technology since its conception. Beyond the first use of cryptocurrencies, it provides many additional features. In the pharmaceutical sector, where reliable supply chains are necessary for cross-border transactions, Ethereum shows promise. It addresses problems through quality, traceability, and transparency in a place defined by complexity and strong laws because of its decentralized structure. As a result, this study looks at how Ethereum is used in the pharmaceutical sector, namely the networks that allow smart contracts to communicate with one another on the Ethereum network. The above concepts are formulated via communication networks, inter-contract owner interactions, and simulation analysis, which seeks to identify dubious practices and unjust contracts inside the supply chain. The study suggests effective manufacturing techniques that call for reduction rather than storage to technological obstacles. With this endeavor, we hope to provide insights into Ethereum-based contract ecosystems and assist in anomaly identification for enhanced security and transparency. The main objective is to support patient record methodology and transform the way healthcare data is managed. The suggested model integrates front-end interfaces, back-end optimization, distributed storage, proof-of-work techniques, and training to establish a safe and efficient ecosystem for healthcare data. These elements can be combined through the blockchain-enabled architecture to transform manufacturing-protecting chemicals in handling, distribution, and necessary training.**

Keywords*—Blockchain; smart contract; externally owned accounts; decentralized trustworthy framework; community healthcare; Ethereum; supply chain management

I. Introduction

Blockchain technology, introduced in 2009 alongside bitcoin, has rapidly transformed many industries beyond cryptocurrencies. The decentralized and secure ledger system of blockchain, first conceived by Satoshi Nakamoto [1], has attracted much interest from industries such as cloud computing, finance, and healthcare maintenance of a distributed ledger that guarantees secure and immutable recording of transactions across a network of nodes, the main objective of this system is that there is no need for centralized control because of the distributed system based on consensus policies by web users called miners.

Among blockchain platforms, Ethereum has emerged as a major player with its introduction in 2015 by Vitalik Buterin [2]. In addition to providing a cryptocurrency (Ether), Ethereum provides a versatile environment for the development of decentralized applications (DAPPs) and the use of smart contracts, encoded as a set of autonomous computer programs Ethereum cornerstone smart contracts facilitate automation, transparency, and efficiency in the blockchain network, which extends its applications beyond simple financial transactions. Understanding the interactions between Ethereum components and smart contracts is critical to unlocking its full potential.

Thus, in this study, blockchain technology provides a solution to ensure end-to-end traceability, traceability, and authenticity of pharmaceutical products throughout their lifecycle. The enabled architecture changes in data management practices by prioritizing patient focus, data security, and regulatory compliance. Analysis of patterns and relationships between blockchains is essential to detect anomalies and ensure the integrity of the ecosystem. The research in [3] outlines an important analytical framework for identifying communication patterns between contracts to classify contracts based on ownership similarity patterns. Overcoming technological challenges is paramount for realizing blockchain's full potential, with significant efforts toward enhancing performance and reducing storage overhead in blockchain data analysis. Therefore, blockchain technology, exemplified by platforms like Ethereum, has transcended its origins in cryptocurrency becoming a catalyst for innovation across diverse sectors. From financial services to healthcare and SCM, blockchain offers a decentralized, transparent, and secure framework for data management, automation, and trustless transactions. Understanding its core principles, deploying smart contracts effectively, and leveraging data analytics tools are essential in harnessing blockchain's trans-formative power and creating value in the digital economy.

The study is as follows—the contribution of the study will be shown in the following section. The background is given in Section III. The related works are discussed in Section IV. The pre-implementation is provided in Section V. The post-implementation is presented in Section VI. The experimental analysis section is out in VII Section. Results are presented in

Section VIII. The conclusion and future works are discussed respectively in Section IX and Section X.

II. CONTRIBUTION

In this contribution section, we look at a microcosm of individual smart contracts participating in exploring the vast landscape and integrating blockchain technology. Our goal goes beyond examining isolated entities, aiming instead to uncover the complex relationships between smart contracts. At the heart of our research is exploring the complexities of the blockchain ecosystem, specifically the interactions between smart contracts. By examining the relationships between these autonomous companies, we seek to identify potential weak spots for common users. The focus of this effort is to analyze contract accounts with their owners, to identify patterns and practices that potentially indicate fraudulent behavior and intention of abuse. Furthermore, apart from observation, our analysis includes a comprehensive analysis of contract renewal issues. There are both challenges and opportunities in the procedure of passing identical contracts on the blockchain. By proper analysis, we can explain the mechanisms of alliance morphology and the causes and consequences for larger ecosystems. Through better analytical methods like machine learning (ML) [4]–[10], we can discover relationships and interactions to better our comprehension of the fundamental processes regulating intelligent transaction agreements. We develop blockchain-based intelligent contract ecosystems by combining these diverse areas of study.

Our findings reveal weaknesses and analogical patterns that provide light on the interactions between smart contracts, offering developers, academics, regulators, and policymakers useful information. In conclusion, our research emphasizes the significance of blockchain technology from a wider perspective; a wide overview of the networks and interdependence that characterize this revolutionary technology, rather than focusing only on individual contracts.

III. BACKGROUND

Two of the primary account types used by Ethereum, the blockchain platform, are contract accounts and external accounts (EOAs). A unique, 20-byte address is assigned to each account, enabling modifications to the status, that include the direct transfer of data and values between accounts. EOAs are not subject to contractual restrictions and are managed by private keys, just like personal bank accounts. Contract accounts, on the other hand, are controlled by their integrated contract rules. Although EOAs are contract accounts, the latter operate independently on the blockchain, allowing each EOA to negotiate or enter into new contracts. Contract accounts can initiate transactions only in response to received transactions—a process referred to in this study as contract-to-contract invokes. Such invokes can trigger diverse actions on the blockchain, including interacting with or executing other contracts and transferring values. Due to their Turing-complete nature, smart contracts can encompass a wide array of functionalities. They may create additional contracts within their code or execute transfers to multiple other contracts. Fig. 1 illustrates an example of a smart contract deployed on Ethereum's blockchain, authored in Solidity. Notably, Solidity version declaration is crucial due to Ethereum's evolving nature, necessitating constant consideration of platform updates and modifications.

```
1  pragma solidity ^0.4.24;
2
3  contract Smartest {
4      mapping (address => uint256) invested;
5      mapping (address => uint256) investBlock;
6
7      function () external payable {
8          if (invested[msg.sender] != 0) {
9              msg.sender.transfer(invested[msg.sender] * (block.number -
                   investBlock[msg.sender]) * 21 / 2950000);
10         }
11
12         investBlock[msg.sender] = block.number;
13         invested[msg.sender]   += msg.value;
14     }
15 }
```

Fig. 1. An Illustration of a smart contract implemented on the blockchain of Ethereum.

The depicted contract, "Smartest", commences with mappings of addresses, storing sender addresses along with associated invested amounts and block numbers. The contract includes a fallback function, an automatic function executed when no other functions match the given identifier. Marked as payable, this function ensures the contract can receive Ether and is externally callable, facilitated by the 'external' modifier. Line 8 verifies if the sender has made any investments, followed by a computation to determine the investment payout based on block numbers. Rapid block addition rate and computation in Ethereum account for approximately 6000 new blocks daily. Consequently, the investment payout, calculated as 4.3%, is dispatched to the investor with subsequent updates performed to the investment mappings and block numbers. In Ethereum, a transaction encapsulates data signed by EOAs for message transmission or contract creation. Transactions originate from EOAs, initiated by signing with their corresponding private keys. Contract accounts interact through messages or internal transactions generated within the Ethereum execution environment. Typically, the sender, identified as the from_address, executes the transaction. EOAs can activate contract accounts, and initiate transactions. In Fig. 2, an EOA triggers a contract-to-contract invoke with a transfer between EOAs. Such invokes often entail specific functions specified in transaction input data, such as transfer and transferEvent, indicating a transfer event within a contract. These functions extract receiver information, either an EOA or a contract.

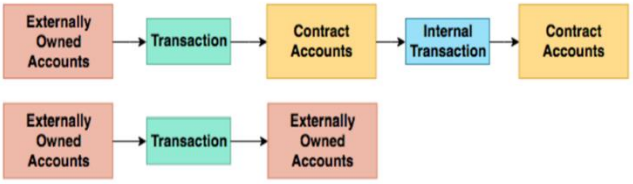

Fig. 2. An Example of a transfer between two EOAs and a contract-to-contract invoke carried out by an EOA.

Contract creation transactions differ by not having a designated receiver address. Instead, they are assigned a unique contract account address linked to the smart contract. Ethereum facilitates the creation of DAPPs, utilizing tokens to represent assets or utilities controlled by smart contracts like ERC20 tokens, which define specific functions applicable to DAPP interactions. Transaction execution incurs operational gas costs, determined by computational requirements to prevent network

abuse. Gas limits, denoted by STARTGAS, curtail resource-intensive actions, ensuring network stability. Miners, validating transactions through proof of work, determine transaction order. Each validated block updates Ethereum's state, forming a blockchain comprising millions of transactions. Web3 Application Programming Interface (API) connectivity enables real-time blockchain interaction, facilitating data retrieval and transaction processing. Infura offers accessible API services for Ethereum connectivity. Utilizing the web3-eth package, developers interact with Ethereum and deploy smart contracts, accessing essential blockchain data like block numbers and transaction details. Scalable data analytics for Ethereum are offered by Google BigQuery, which provides blockchain data for effective Structured Query Language (SQL) querying. Another perspective can be obtained by analyzing data from the Ethereum blockchain encoded in other formats, such as JavaScript Object Notation (JSON) and Avro. However, processing big data requires more effective data filtering techniques, especially given Ethereum's expanding size. Due to this rapid user growth, Ethereum has become particularly rich. Its blockchain has grown to over 500 GB and is still quickly expanding. Efficient methods for handling these large datasets become important when dealing with such large amounts of data. Understanding the nuances of the memory architecture shown in Fig. 3 is important for proper system performance. Different levels of memory are different in a hierarchical structure and each affects how quickly data can be accessed. L6 or secondary remote storage, like shared file systems and web servers, is the lowest. According to [11], there is a noticeable performance delay in receiving data from L6. On the other hand, Random Access Memory (RAM) storage in layers L2 to L4 provides faster data access and consequently improves performance. Therefore, understanding and implementing these memory systems is essential for the system to work properly in Ethereum blockchain channels.


```
{'hash': '0xecfee04e3c04490bdac99ea00b75bd620f7a36e9f6b9919d47e01376644233bd',
'nonce': '0',
'transaction_index': '118',
'from_address': '0x91d0893509f3bd4271b2106f94de51dfd459edea',
'to_address': '0x0bd57fc1289c2b8e090e5f2f7d972514e6a494fb',
'value': 0,
'gas': '57016',
'gas_price': '18000000000',
'input': '0xa9059cbb000000000000000000000000956ead78bd38d39d486a058a22cf35334
          1fc3a90000000000000000000000000000000000000000000000021e19e0c9bab2400000',
'receipt_cumulative_gas_used': '5461853',
'receipt_gas_used': '22016',
'receipt_contract_address': None
'receipt_status': '1',
'block_timestamp': '2019-02-25 09:14:30 UTC',
'block_number': '7265339',
'block_hash': '0xd667a2d324f8122f4f5f15bd1669a68ba3a519150c648d4b1df72e4236efbb69'}
```


Fig. 3. A single transaction of Ethereum data in the big query.

IV. Related Work

Smart contracts, the core of blockchain technology, have been the subject of many studies recently, including the work of [12] to optimize development processes and address security flaws, a comprehensive study shows specific issues faced by those working with Solidity, the famous company programming language for blockchain-based systems made, one of the outstanding contributions in this regard is [13] described some security issues with Solidity smart contracts, such as a well-known reentry attack with the ability to compromise the entire integrity-at-risk contract system and launched several programs aimed at improving it. The emergency stop measure is one such model that can effectively reduce the risks associated with the conclusion of criminal contracts. The study in [14] as a result of the project, flexible and secure smart contracts were developed, providing effective responses to these security issues. The research in [15] also made a notable addition—when they developed SmartInspect, a system that facilitates editing and visualization of individual smart contracts SmartInspect allows developers to graphically represent contract code and additional special rules or conventions in an imageless system such as Ethereum. The debugging process can be accelerated without reuse, where data is stored as bytecode. By simplifying the development process and improving smart contract debugging, this tool ultimately increases the common dependencies on blockchain-based systems in a new way of managing Solidity. The research in [16] solved the tricky speed, a journey was started using the Smalltalk Compiler Compiler (SmaCC). to develop a parser compatible with Pharaoh of programming environment This effort was based on the need to guide Solidity, the leading language in smart contract development, through its inherent errors and ambiguities. By carefully building a parser it gets to the details of Solidity aimed at re-moving complexity and paves the way for advanced debugging and security measures in smart contracts.

A careful analysis of Solidity's [17] grammar and semantics was a key factor in the work. By analyzing the structure of the language, they were able to identify major barriers to its exploration. These challenges had problems, such as grammar inconsistencies and simple formulas, which led to frequent parsing errors and hindered proper understanding Solutions were developed to alleviate these challenges through in-depth analysis and over iterative development cycles, and a robust parsing tool for intelligent and secure contracts [18] is one of his most important contributions. Due to their autonomous and invariant rules, smart contracts require high accuracy and reliability when implemented. However, due to the physiological complexity of languages such as Solidity, there are major obstacles to achieving this goal. The study in [19] strengthens the security posture of smart contracts by improving their understanding and debugging with a parsing

customized to Solidity's peculiarities. The study of [20] establishes how blockchain technology has interdisciplinary emphasis, compilers for solving practical problems, computer language, etc. By integrating knowledge from other industries and applying Smalltalk ecosystem tools and methodologies, they demonstrate the value of interdisciplinary approaches in promoting blockchain development. The relevance of the study [21] goes beyond the research methods. It highlights a larger trend in smart contract research where researchers and industry partners collaborate to bolster blockchain technology initiatives. The researchers worked together to find security flaws and provide reliable development tools, accelerating smart contracts and establishing them as key features of a decentralized app. Table I shows the summary of the above literature.

TABLE I. SUMMARY OF THE RELATED WORKS

Work	Optimized Development	Addressed Security Flaws	Improved Debugging	Enhanced Parsing	Strengthened Security	Interdisciplinary Approach	Collaboration with Industry
[12]	✓	✓					
[13]		✓			✓		
[14]					✓		
[15]			✓				
[16]				✓			
[17]				✓			
[18]				✓			
[19]				✓	✓		
[20]				✓		✓	
[21]						✓	✓

V. PRE-IMPLEMENTATION

This section explores various analytical approaches by testing the necessary assumptions and fitting the appropriate methods. Data storage design specifications are also described to aid understanding and use. Our main goal is to divide Ethereum transactions into four categories—token transactions, ether transfers, contract creation, and contract-to-transaction calls. The categories above allow for a thorough examination of the behaviors and network patterns inside the Ethereum ecosystem. The attributes required for database inclusion have been determined to classify contracts by ownership containing the contract code, owner address, contract address, nonce, cost, and timestamp. These datasets are available in JSON format with a size of 2GB as shown in Table II. Using BigQuery and Web3 API, many datasets with particular features are pulled from Ethereum to enable thorough studies of Ethereum blockchain operations. For example, the Owner Address—which identifies the EOAs that created the contract—the Contract Address—which is a unique identifier for the newly created contract account—and the Contract Code—which is essential for calculating a hash value to distinguish identical copies of contracts—are required in the first dataset, which is focused on Contract Creations. It's also crucial to include the Nonce attribute, which shows how many contracts have been created by a particular account; the Value attribute, which shows asset flow for upcoming asset flow analysis; and the Timestamp, which gives the temporal context for contract creations. Important elements of the invoke dataset include entering the Contract-to-Contract Phase Aspects, such as the Owner Address (relating to the EOA initiating the transaction), the Contract Address (Sender), which is the contract's address initiating the function call, and the Receiver Address, which is the contract's or EOA's address receiving the function call or transfer. For important in-depth research, the Receiver Type checking (EOA or contract account), Input Data with bytecode with some function call statement, Nonce for tracking transactions, Asset value of contracts transfers, and Timestamp for context identification are required. Therefore, in the Ether transfer dataset, the final properties should be the Owner Address (to initiate transactions), Receiver Address (EOA or contract account), Receiver Type checking, Nonce for transaction counts, Asset value movement, and Timestamp for the context of transactions. We can analyze contract behaviors, track asset flows, and recognize temporal behavior transactions to measure the Ethereum ecosystem's quality. Therefore, by storing these datasets in either NoSQL or relational databases in the ecosystem, query performances can be improved and analytical procedures can be developed apart from maintaining the data integrity.

Every characteristic plays a distinct part in separating behavioral results from ownership ties. Consequently, relevant information such as owner address, contact address (sender and receiver), data entry, nonce, price, and timestamp are identified for contract-to-contract calls. These characteristics make it easier to examine transactional exchanges and the movement of assets between contracts in greater detail. For completing Ether transfers, variables like possessor location, receiver address, nonce, sum, and time mark are required for detecting transactional dynamics and asset movements. These characteristics ensure a complete understanding of transactional behaviors and open doors for further analysis. As mentioned in the background section, there are two main approaches to extracting Ethereum blockchains: BigQuery and Web3 API. The benefits and losses of each technique are carefully considered to support the decision-making process. BigQuery is a useful tool for extracting large amounts of data; it can export the full blockchain dataset. However, it has limitations, such as storage requirements and limited scalability.

Conversely, Web3 API provides immediate access to the most recent blockchain data, although it might rely on external APIs and encounter performance challenges. Subsequently, the focus transitions to clustering analysis, wherein transactional models and behaviors are de-ciphered utilizing unsupervised learning methodologies [22]–[27]. The effectiveness of the k-means clustering method is emphasized in terms of its ability to divide data into discrete groups according to similarities. The ideal number of clusters can be found using techniques like the Elbow method. It makes it easier to analyze and comprehend transactional data meaningfully.

However, to facilitate effective data retrieval and analysis, the database architecture also attempts to create a hierarchical structure that arranges contracts according to ownership connections. Therefore, a thorough approach to transactional behavior analysis and database design is also considered, laying the foundation for data extraction, analysis, and interpretation inside the Ethereum ecosystem. However, pre-processing is necessary to build an exhaustive invokes tree of contract-to-contract calls. Additional data sorting and storage are part of this phase. The main goal is to classify Ethereum transactions into three main categories—ether transfers, contract-to-contract invokes, and contract constructions, as was previously mentioned. Separating Tokens according to the kind of transaction they involve i.e., differentiating between contract-to-contract invokes and normal transfers—is another essential goal. This preparatory phase lays the foundation for the next examination and knowledge of the complex dynamics of Ethereum transactions. Important information about transaction hashes and Ethereum token contract addresses are kept in the BigQuery file section called token transfers. A complete inventory of all tokens requires a preliminary preprocessing step that includes a thorough review of all token-transfers files and a methodological filing of every contract address into a separate file. Similarly, creating an exhaustive Token list that lists every contract requires a preprocessing phase that involves going through every transaction file and methodologically storing every contract address linked to a contract establishment into a different file. This initial step is necessary since account addresses are all the same, regardless of whether they are contract accounts or EOAs, and they are all 20-byte hexadecimal addresses that are not unique from one another. An important phase of this process is distributing distinct among EOAs and contract accounts, which requires a detailed verification process facilitated by the above list.

TABLE II. DATASET DESCRIPTION

Dataset Name	Data Categories Included	Required Attributes	Size (JSON Format)	Storage Type	Purpose/Analysis Focus
Contract Creations	Contract Creation Transactions	Owner Address, Contract Address, Contract Code, Nonce, Cost, Timestamp	2GB	NoSQL/Relational	Analyzing contract creation behaviors and patterns
Contract-to-Contract Invokes	Contract-to-Contract Interaction Transactions	Owner Address (Initiator), Contract Address (Sender), Receiver Address, Receiver Type, Nonce, Asset Value, Timestamp	2GB	NoSQL/Relational	Studying interactions and asset flows between contracts
Ether Transfers	Ether Transfer Transactions	Owner Address (Initiator), Receiver Address, Receiver Type, Nonce, Asset Value, Timestamp	2GB	NoSQL/Relational	Analyzing asset movements and transaction dynamics
Token Transfers	Token Transfer Transactions	Transaction Hash, Ethereum Token Contract Addresses	BigQuery File	BigQuery	Tracking token transactions and contract addresses
Contract Inventory	Ethereum Contract Inventory	Contract Addresses	BigQuery File	BigQuery	Maintaining a record of all contract addresses on Ethereum
Token List Inventory	Ethereum Token List	Contract Addresses (Tokens)	BigQuery File	BigQuery	Creating a comprehensive list of all Ethereum tokens

VI. POST-IMPLEMENTATION

This section outlines the framework architecture, including the design and implementation process. A pipe and filter design defines the general architecture of the framework, as seen in Fig. 4.

Black pumps on the left side of the diagram stand in for all of the transactions and token-transfer data are obtained from the Ethereum blockchain. A single usage of these token-transfer files is made to create an exhaustive list of token contract addresses. On the other hand, there are numerous uses for the transaction files. First, they make it easier to create a file on the blockchain to have all the active contract addresses. They are then passed through the Contract Creator (CC) filter responsible for classifying contracts according to their owners and performing hash calculations to obtain hash strings for every contract. Every transaction is thoroughly processed by the CC filter, which also performs verifications against the contract address file and extracts pertinent information. The resulting data is stored in a CC Database and consists of owners grouped with their corresponding contracts. This dataset is put through one extra round of filtering, to create visuals that clarify the connection between owners and contracts. In the context of transaction file utilization, another significant application arises—the sorting and processing of transactions excluding contract creations. This specific filtration process is termed Invokes Creator. It operates simultaneously with lists of contracts and token contract addresses for verification purposes. Initially, the preprocessing stage segregates transactions into three databases—tokens, transfers, and a

database labeled calls, housing all contract-to-contract transactions [28]. This methodology facilitates efficient management and organization of transactional data, ensuring systematic handling of diverse transaction types.

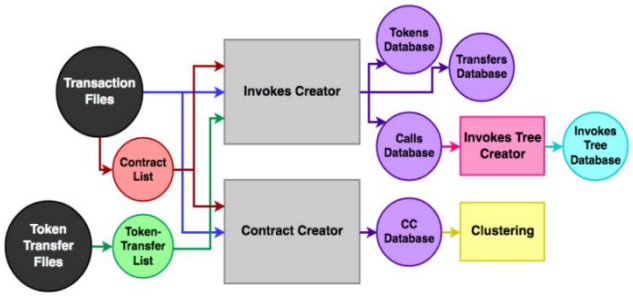

Fig. 4. The framework's architecture.

The investigation brought to light the considerable amount of data being processed. A sequential data processing methodology is embodied in the architectural framework shown in Fig. 5. However, this step-by-step approach can considerably increase the processing time. As such, the next part offers a paradigm change in the direction of parallel execution. The example process described in this section involves splitting the dataset into N segments, each one to be processed by a separate processor. The purpose of this parallel processing technique is to reduce processing time and maximize the use of computational resources. However, a significant part of the process, approximately 0.58% of all transactions are contract creations. Therefore, this procedure should ideally be run on a single CPU for efficiency.

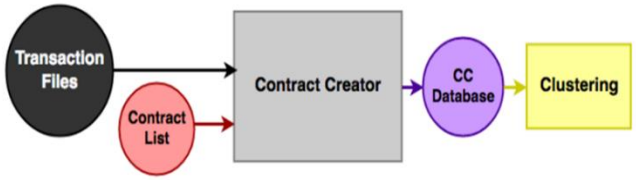

Fig. 5. The procedure for carrying out the gathering of contract creations and grouping.

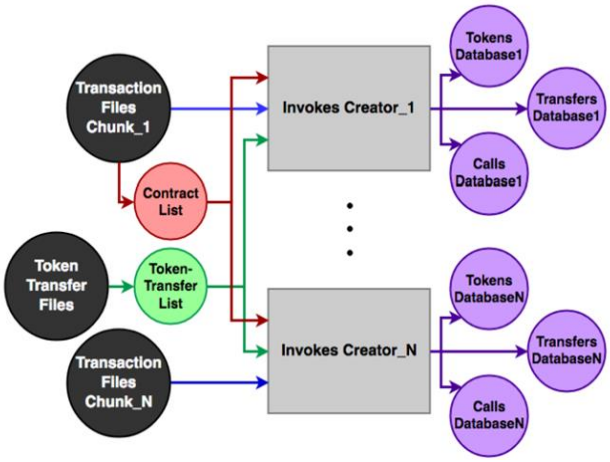

Fig. 6. The technique of contract-to-contract calls/ transfers/tokens execution.

Subsequently, we investigate how Invokes Creator extracts calls, transfers, and tokens between contracts. This crucial step in data analysis takes longer because a lot of data is involved. As shown in Fig. 6, the process can be divided into smaller components, resulting in multiple databases. After that, Invokes Tree Creator processes each dataset and builds an extensive tree that displays every invoke between contracts. This systematic procedure facilitates a thorough analysis of the network's interactions.

A. Additional Improvements

HDF5 [29] is a better alternative to current database architecture, with a hierarchical database structure suitable for managing large datasets. It is compatible with programming languages, like Fortran, Octave, Mathematica, Scilab, MATLAB, R, Julia, and Java. One of its main advantages is more efficient use of storage resources than conventional relational databases. Its compact format reduces processing costs and file space usage and is useful for tiny datasets. It is possible to duplicate this hierarchical structure in a relational database but may add redundancy, which raises storage requirements. Additionally, more complex query operations like invokes are required in relational databases to extract certain subtrees. However, HDF5 has a restriction of performing only one read or write operation at a time.

Contrary to that, the concurrency characteristics of relational databases allow several simultaneous reads and writes due to their transaction-savvy design. The choice of language is crucial when creating a framework with wide application. Among the languages supported by HDF5, Python stands out due to its extensive library ecosystem. Although Python is an interpreted language, meaning it typically performs less computationally efficiently than compiled languages—this disadvantage becomes less significant when considering the dominating disk activities that occur during runtime. As such, Python's alleged processing latency may not exceed its advantages. Multi-threading is replaced by Python's global interpreter lock, though, unless it is augmented by other tools that function outside of Python's domain, like non-Python libraries or network requests. The h5py Python library extension provides Python access to HDF5 features by treating h5py datasets as NumPy arrays and h5py groups as Python dictionaries. The implementation of Python and HDF5 for best speed presented issues because of HDF5's single-threaded read/write capability and Python's lack of multithreading support. Multithreading was first used to investigate network requests—however, this method proved unstable because of frequent failures and time-outs. Thus, the search was on for a network-independent solution. The best method for reaching peak performance was to spread data among the available servers each running five to six processes to exploit hardware resources. However, system stability was crucial because high thread utilization might cause crashes, particularly when the processes ran under NFS from a notorious home folder for its sporadic instability. As a result, keeping processes to 5–6 allowed for a compromise between speed and stability. Managing databases was an essential component of the finished system. For data sorting, 153 databases were constructed (not including Contract Creations) in light of HDF5's single read/write constraint.

VII. Experimental Analysis

During the first part of our study, we processed large-scale datasets that were taken out of BigQuery and used the Web3 API to confirm the addresses' validity. This made it necessary to distinguish between contract accounts and EOAs. We chose to implement multi-threading in Python to maximize efficiency; this choice was influenced by the computing demands of our task and the limitations imposed by Python's global interpreter lock. We created a three-threaded pipeline to expedite the data pre-processing step to demonstrate our methodology as shown in Fig. 7.

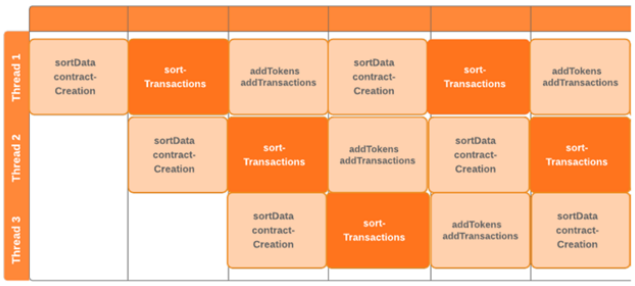

Fig. 7. The first solution is a pipeline through the network.

Components in this pipeline indicated by dark orange boxes are used to classify transactions that do not include CC. The performance of a proposed solution heavily relies on its efficiency, particularly evident in systems utilizing the Web3 API. As an empirical evaluation, the initial approach exhibited suboptimal results due to extensive network connections. During time experiments with modest data sets comprising three small files of 2.42 MB, a stark contrast in performance emerged. For preprocessing, the preliminary solution necessitated a substantial 340.825 seconds while the optimized solution devoid of network interactions accomplished the same task in a mere 11.093 seconds, boasting a remarkable speed-up factor of 31. Nonetheless, it is imperative to acknowledge that the optimized solution introduces its preprocessing phase. This entails traversing through transaction files to aggregate all generated contract addresses. Furthermore, an intermediary enhancement step contributed to the overall acceleration. Initially, the program loaded all contract addresses into a list. However, transitioning this data structure to a dictionary yielded notable improvements. The processing time is reduced from 259.207 to 16.782 seconds, demonstrating a notable speed-up factor of 15. The results of using the suggested technique showed promise in runtime efficiency. After two and a half hours, the preparation operation was finished, processing a large dataset of 265 GB. This accomplishment highlights how well the distributed processing strategy handles a massive volume of data.

VIII. Result Analysis

When using blockchain systems in real-world applications across different industries, scalability, and efficiency are critical factors to be considered. The performance of the underlying hardware becomes a bottleneck for transaction processing and examination, where a significant volume of transactions may be involved. As per the given data, it has taken up to 9.5 hours to process and comment for the 408,137,399 transactions via the hardware as illustrated in Table III. The number of transactions highlights the requirement for a strong hardware infrastructure to manage these processing demands. Clustering and creating the Invokes tree are two computationally intensive processes; considered elements of the inspection phase detailed in the data. Clustering is assembling transactions based on many parameters, such as transaction type, origin, or destination to understand transaction trends and behaviors. Significant computer resources are needed for this procedure, especially when working with huge datasets like the one mentioned. The trade-off between computational complexity and hardware capability is shown in the execution time of 9.5 hours for processing and reviewing over 400 million transactions. Although huge workloads may be handled by modern hardware, processing massive datasets quickly is challenging, especially in distributed and decentralized contexts where resource limitations and network delay are issues.

TABLE III. Performance on Hardware

Hardware Configuration	Transactions Processed	Execution Time
Intel Xeon Gold 6248	408,137,399	9.5 hours

A. Clustering Analysis

The results shown in Fig. 8, 9, and 10 on the clustering equivalency classes among owners with an equal number of contracts are the focus of the analysis in this section. The Elbow method, which is explained, is applied in Fig. 8 and 9. In particular, Fig. 8 illustrates the Elbow method's use throughout a k range of up to 15, showing a clear bend in the graph around the point where three clusters correspond. Similarly, Fig. 9 presents the Elbow method utilizing a k range up to 25, wherein the inflection point aligns with the presence of three clusters. Consequently, it can be inferred that the optimal number of clusters is three. Subsequently, Fig. 10 visually presents the clustering outcome with three clusters, with the centroids depicted as black spots. The clustering analysis indicates variations in the distribution of EOAs based on contract frequency. It reveals that certain EOAs exhibit a high concentration of contracts, while others demonstrate a more dispersed pattern with increased occurrences. Notably, a significant portion of EOAs falls within the green cluster, indicating a range of approximately 1 to 1000 contracts per EOA.

Similarly, Fig. 11 and 12 illustrate the outcomes of the analysis. Fig. 11 depicts the application of the Elbow method within a k range extending to 15, revealing a pivotal bend in the graph around three clusters. This methodology was further extended to k values up to 25, yet consistent with prior findings, the optimal number of clusters remained at three. The analysis depicted in Fig. 12 highlights the prevalence of numerous distinct contracts associated with each EOA, primarily represented by the purple segment. However, a notable observation emerges from the red cluster, indicating instances where individual proprietors possess over 100,000 identical contracts. An escalation in the number of contracts and the distribution shifts indicate a transition towards a multitude of unique contracts or a proliferation of duplicates.

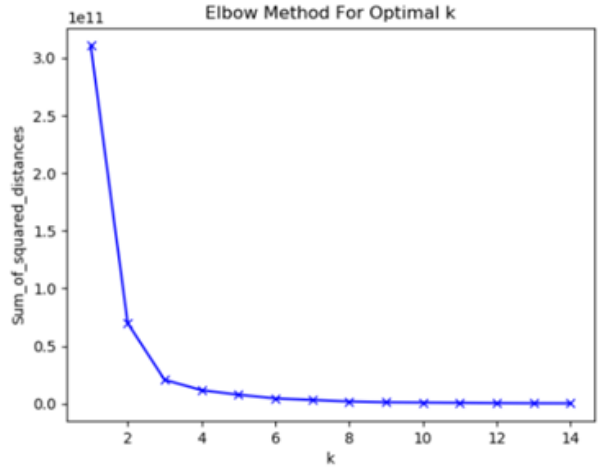

Fig. 8. Using k in the range of 15, determine the appropriate number of clusters for the range of contracts that occur each EOA concerning the number of times that this amount occurs.

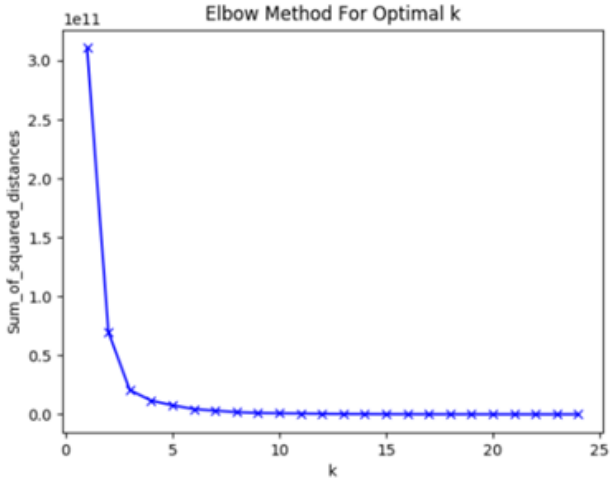

Fig. 9. Using k in the range of 25, determine the appropriate number of clusters for the range of contracts that occur every EOA concerning the number of times this amount occurs.

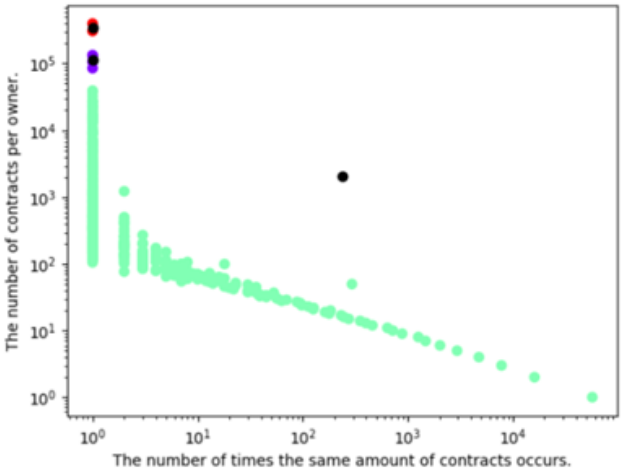

Fig. 10. The distribution of contracts per EOA compared to how frequently this amount occurs, shows three clusters.

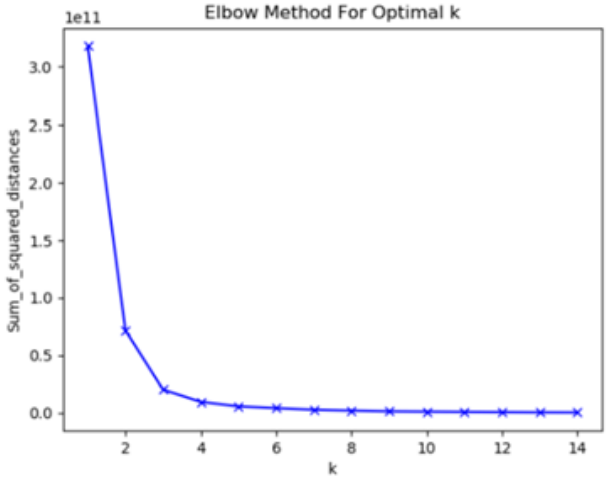

Fig. 11. Using k in the range of 15, determine the appropriate number of clusters for the number of contracts per EOA with the number of unique contracts per EOA.

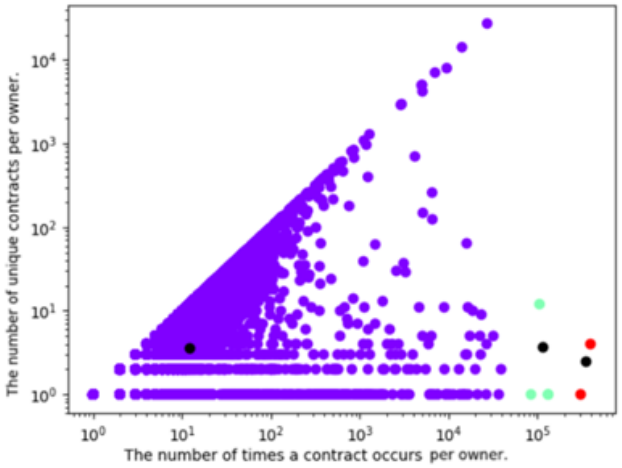

Fig. 12. Using three clusters, the unique number of contracts per EOA is clustered with the total number of contracts per EOA.

Additionally, Fig. 13 and Fig. 14 delve into a comparative examination concerning the recurrence of contract instances and the number of distinct EOAs utilizing each contract. Fig.13 employs the Elbow method across a range of k values up to 15, elucidating that three emerge as the optimal cluster count. Subsequent iterations of this method reaffirm the consistency of three as the most favorable cluster count. In the clustering analysis illustrated in Fig.14, we examine the distribution of contract duplications across various Ethereum-based organizations. Notably, the initial instance of an EOA is excluded from consideration, allowing us to observe subsequent EOAs adopting a contract already in circulation. The depiction highlights a discernible pattern—a distinct concentration of contracts utilized by multiple EOAs, represented by the purple area, alongside a cluster of contracts scarcely replicated by EOAs. This observation suggests a dual inclination among EOAs, favoring either widely duplicated contracts or those with minimal replication.

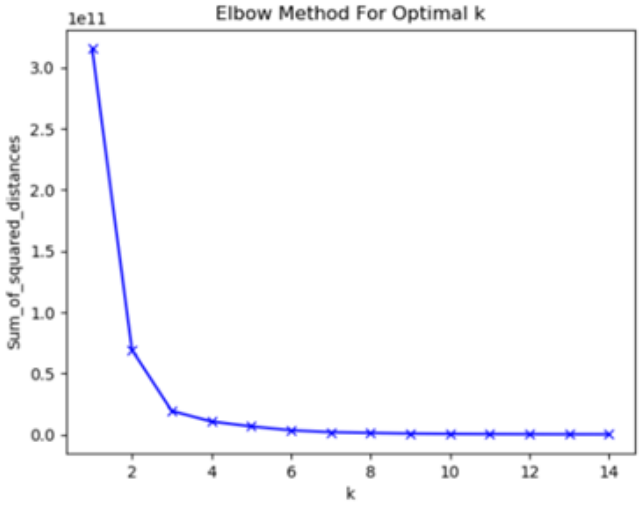

Fig. 13. Using k in the range of 15, get the number of clusters for the number of times a contract happens more than once using the number of unique EOAs using each contract.

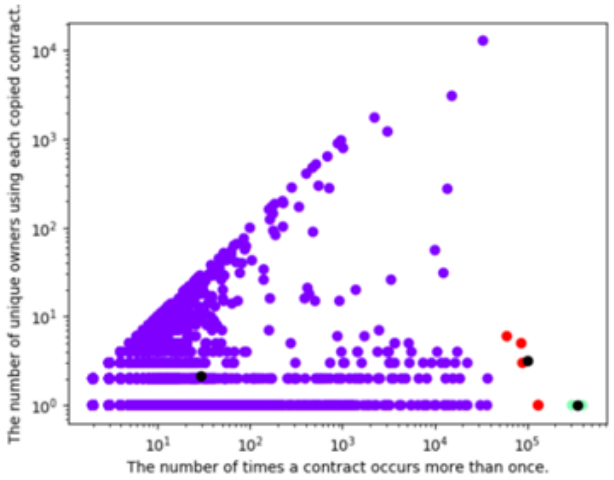

Fig. 14. Three clusters used to group the number of times a contract happens more than once and the number of distinct EOAs that use each contract.

B. *Invokes Tree Analysis*

Analyzing the execution of smart contracts is essential for locating any weak points and illegal activity in blockchain networks. A structured network of invokes is revealed in the performed examination, providing insight into the complex relationships between different smart contracts. This network is illustrated graphically in Fig. 15, where the contract code for the address Green Ethereus is highlighted along with its interactions with three different contracts—SuperFOMO, UCashBank, and Smar-tHash. However, a closer look at the SuperFOMO contract, reveals more calls than were first thought to be there, including exchanges with Gorgona, EtherSmart, and self-referential messages. A hierarchical pattern of invokes similar to a tree-like structure is shown in this study, describing the relationship complexity within smart contracts. A closer verification of the contract addresses in the invokes tree clarifies that there are questions about the true nature of the root contract, Green Ethereus. Green Ethereus operates more like a Ponzi scam, using its ties to other contracts to perpetrate more fraud. Furthermore, a thorough examination of each contract that Green Ethereus has cited explores characteristics common to Ponzi schemes, emphasizing the interconnectedness of the fraudulent activity inside the network. These findings demonstrate the significance of the Invokes Creator's role in avoiding circular reference connections in contracts. The blockchain network, as a whole, may become vulnerable to vulnerabilities introduced by circular references, jeopardizing its integrity and security.

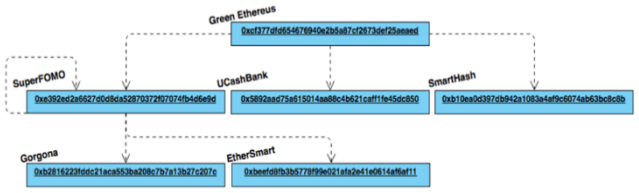

Fig. 15. An illustration of a 'Transaction Tree' from the generated data that combines multiple confirmed Ponzi schemes.

To improve readability and avoid misunderstandings when displaying invokes, the existing method suggests repeating the contract address as a sub-group, as seen in Fig. 16. This approach makes it easier to comprehend how smart contracts interact by providing a simple enhancement to the visualization tool. It should be noted that time restrictions prevented this approach from being implemented, but only highlighting the practical issues that must be critical. The iterative and resource-constrained nature of blockchain is reflected in the decision to prioritize features or performance enhancements. The sequence of essential capabilities, time constraints, and technical limitations often guides the execution trajectory of a study. In this instance, the suggested course of action might be clearer and easier to understand than the others; the other urgent issues might have prevented it from being put into action immediately. Besides, potential Ponzi schemes and fraudulent behaviors within the blockchain network urge the development of scrutiny and supervision protocols.

Fig. 16. Current invokes tree cyclic reference solution.

Continuous monitoring and subsequent preventative actions are required to ensure the reliability and authenticity of blockchain-based systems. Participants should effectively minimize the adverse impacts of fraudulent activities to provide a more resilient and secure blockchain ecosystem by applying insights gained from smart contract analysis and proactive risk mitigation strategies. Thus, the intricate invoke network facilitating smart contract communication is made visible by incorporating it into the smart contract implementation process.

The discovery of fraudulent activity and Ponzi schemes highlights the strength and usefulness of blockchain networks. Although suggested solutions by the Invokes Creator are beneficial to reduce risk, real concerns can be challenging in practical situations. Proactive security and risk management are required to maintain the integrity and reliability of blockchain systems.

IX. Conclusion

The attainment of goals for the research work is rigorously tried. The goals are achieved, leading to discoveries that open new avenues for blockchain data research and analysis. The proposed framework and the obtained outcomes demonstrate that the objectives are met. Firstly, collecting a required quantity of well-organized data is an important task. The dataset contains extra categories that are not directly related to the study. The analysis of categorized token data and Ether transfers can provide trends and insights on blockchain activity that do not fall within the scope of the study, allowing further investigation and analysis. Moreover, the clustering data reveal both normal and aberrant equivalence groups of owners with identical contract numbers, offering compelling proof of contract repetitions. This discovery advances our understanding of blockchain dynamics and highlights the significance of locating and analyzing contract replication patterns inside the network. Lastly, the invokes tree exposes important distinctions between contract-to-contract invocations, exposing in-stances such as pyramid schemes. It is acknowledged that chronological constraints have limited the fullness of the CC tree. In particular, it is difficult to fully capture the scope of contract interactions when contracts are not included as sub-groups to other contracts. This restriction is due to the way contracts are created; which forces the sender to default to the EOAs whether or not the contract was created through the instantiation of another contract. Despite this problem, possible ways to address the disparity are suggested. This constraint could be addressed by doing checks against each contract account nonce and combining tree patterns from the invokes tree to provide a more complete picture of contract interactions. Crucially, it is seen that this structural problem does not interfere with the clustering process, allowing for further investigation even in the absence of an instant answer.

X. Future Work

Our research extends the existing framework to incorporate the latest blocks from the Ethereum blockchain. Presently, the framework holds data up to March 18, 2019. This augmentation facilitates the gathering and processing of up-to-date blockchain information. The clustering analysis illustrates discernible patterns within the data. Notably, the identified clusters consist of three distinct groups. An intriguing avenue for future investigation involves conducting clustering analyses on the subgroups within these clusters to unveil deeper insights into underlying patterns and structures. We acknowledge the existence of platforms cataloging registered scams within the Ethereum ecosystem. Promising future research entails scrutinizing these scam patterns to develop a predictive model to pre-emptively identify potential future scams. Our structure carefully arranges information, providing a rich environment for studying transfers among EOAs. This research can potentially identify Ether's flow and stall in the network. Moreover, our research can extend the Invokes tree to incorporate links revealing cycles within the blockchain ecosystem. An integral aspect of our research involves developing an extension that traverses through each database, elucidating the flow of Ether, colloquially termed as following the money. This comprehensive analysis of the movement of Ether among contract-to-contract involves Ether transfers. Additionally, our investigation entails juxtaposing our findings with the dates of well-known attacks, enriching our understanding of the security landscape within the Ethereum ecosystem.

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

Data is available on request from the corresponding author.

Acknowledgment

The authors extend their appreciation to the Deanship of Scientific Research at Saudi Electronic University for funding this research (8154).

References

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” SSRN Electronic Journal, 2022, doi: 10.2139/ssrn.3977007.

[2] Buterin and Vitalik, “Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform,” Etherum, no. January, pp. 1–36, 2014, Accessed: April 07, 2024. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper.

[3] M. E. Peck, “Blockchains: How they work and why they’ll change the world,” IEEE Spectrum, vol. 54, no. 10, pp. 26–35, Oct. 2017, doi: 10.1109/MSPEC.2017.8048836.

[4] M. K. I. Rahmani et al., “Blockchain-Based Trust Management Framework for Cloud Computing-Based Internet of Medical Things (IoMT): A Systematic Review,” Computational Intelligence and Neuroscience, vol. 2022, p. 18, 2022.

[5] M. K. I. Rahmani et al., “Automatic Real-Time Medical Mask Detection Using Deep Learning to Fight COVID-19,” Computer Systems Science and Engineering, vol. 42, no. 3, pp. 1181–1198, 2022.

[6] S. Safdar et al., “Bio-Imaging-Based Machine Learning Algorithm for Breast Cancer Detection,” Diagnostics, vol. 12, no. 5, p. 1134, 2022.

[7] N. Awan et al., “Machine learning-enabled power scheduling in IoT-based smart cities,” Computers, Materials & Continua, vol. 67, no. 2, pp. 2449–2462, 2021.

[8] M. A. Khan et al., “Investigation of Big Data Analytics for Sustainable Smart City Development: An Emerging Country,” IEEE Access, vol. 10, pp. 16028-16036, 2022.

[9] P. Kaur, G. S. Kashyap, A. Kumar, M. T. Nafis, S. Kumar, and V. Shokeen, “From Text to Transformation: A Comprehensive Review of Large Language Models’ Versatility,” Feb. 2024, Accessed: Mar. 21, 2024. https://arxiv.org/abs/2402.16142v1

[10] K. Bhalla, D. Koundal, S. Bhatia, M.K.I. Rahmani, and M. Tahir "Fusion of Infrared and Visible Images Using Fuzzy Based Siamese Convolutional Network," Comput. Mater. Contin., vol. 70, no. 3, pp. 5503-5518. 2022.

[11] R. E. Bryant et al., “Computer systems: a programmer’s perspective,” p. 1043, 2011, Accessed: May 07, 2024. [Online]. Available: https://thuvienso.hoasen.edu.vn/handle/123456789/8621.

[12] M. Wohrer and U. Zdun, “Smart contracts: Security patterns in the ethereum ecosystem and solidity,” in 2018 IEEE 1st International Workshop on Blockchain Oriented Software Engineering, IWBOSE 2018

- Proceedings, Mar. 2018, vol. 2018-Janua, pp. 2–8. doi: 10.1109/IWBOSE.2018.8327565.

[13] J. F. Ferreira, P. Cruz, T. Durieux, and R. Abreu, "SmartBugs: A Framework to Analyze Solidity Smart Contracts," in Proceedings - 2020 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Sep. 2020, pp. 1349–1352.

[14] T. Krupa, M. Ries, I. Kotuliak, K. Košál, and R. Bencel, "Security issues of smart contracts in ethereum platforms," in Conference of Open Innovation Association, FRUCT, Jan. 2021, vol. 2021-Janua.

[15] S. Bragagnolo, H. Rocha, M. Denker, and S. Ducasse, "SmartInspect: Solidity smart contract inspector," in 2018 IEEE 1st International Workshop on Blockchain Oriented Software Engineering, IWBOSE 2018 - Proceedings, Mar. 2018, vol. 2018-Janua, pp. 9–18. doi: 10.1109/IWBOSE.2018.8327566.

[16] H. Rocha, S. Ducasse, M. Denker, and J. Lecerf, "Solidity parsing using SmaCC: Challenges and irregularities," in IWST 2017 - Proceedings of the 12th International Workshop on Smalltalk Technologies, in conjunction with the 25th International Smalltalk Joint Conference, Sep. 2017. doi: 10.1145/3139903.3139906.

[17] J. Jiao, S. Kan, S. W. Lin, D. Sanan, Y. Liu, and J. Sun, "Semantic understanding of smart contracts: Executable operational semantics of solidity," in Proceedings - IEEE Symposium on Security and Privacy, May 2020, vol. 2020-May, pp. 1695–1712. doi: 10.1109/SP40000.2020.00066.

[18] Z. Wang, X. Chen, X. Zhou, Y. Huang, Z. Zheng, and J. Wu, "An Empirical Study of Solidity Language Features," in Proceedings - 2021 21st International Conference on Software Quality, Reliability and Security Companion, QRS-C 2021, 2021, pp. 698–707. doi: 10.1109/QRS-C55045.2021.00105.

[19] Á. Hajdu and D. Jovanović, "SOLC-VERIFY: A modular verifier for solidity smart contracts," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2020, vol. 12031 LNCS, pp. 161–179. doi: 10.1007/978-3-030-41600-3_11.

[20] I. Garfatta, K. Klai, W. Gaaloul, and M. Graiet, "A Survey on Formal Verification for Solidity Smart Contracts," in ACM International Conference Proceeding Series, Feb. 2021. doi: 10.1145/3437378.3437879.

[21] S. W. Lin, P. Tolmach, Y. Liu, and Y. Li, "SolSEE: a source-level symbolic execution engine for solidity," in ESEC/FSE 2022 - Proceedings of the 30th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Nov. 2022, pp. 1687–1691. doi: 10.1145/3540250.3558923.

[22] M. S. Islam, M. A. B. Ameedeen, M. A. Rahman, H. Ajra, and Z. B. Ismail, "Healthcare-Chain: Blockchain-Enabled Decentralized Trustworthy System in Healthcare Management Industry 4.0 with Cyber Safeguard," Computers, vol. 12, no. 2, p. 46, 2023.

[23] S. B. Khan et al., "Artificial Intelligence in Next-Generation Networking: Energy Efficiency Optimization in IoT Networks Using Hybrid LEACH Protocol," SN Computer Science, vol. 5, no. 5, p. 546, 2023.

[24] L. Stockburger et al., "Blockchain-enabled decentralized identity management: The case of self-sovereign identity in public transportation," Blockchain: Research and Applications, vol. 2, no. 2, 2021.

[25] Y. F. Khan et al., "HSI-LFS-BERT: Novel Hybrid Swarm Intelligence Based Linguistics Feature Selection and Computational Intelligent Model for Alzheimer's Prediction Using Audio Transcript," IEEE Access, vol. 10, pp. 126990-127004, 2022.

[26] M. A. Khan et al., "Artificial Intelligence in Commerce and Business to Deal with COVID-19 Pandemic," Turkish Journal of Computer and Mathematics Education (TURCOMAT), vol. 12, no. 13, pp. 1748-1760, 2021.

[27] M. K. I. Rahmani, N. Pal, and K. Arora, "Clustering of Image Data Using K-Means and Fuzzy K-Means, (IJACSA) International Journal of Advanced Computer Science and Applications, vol. 5, no. 7, 2014.

[28] Arwa Mukhtar, Awanis Romli and Noorhuzaimi Karimah Mohd, "Blockchain Network Model to Improve Supply Chain Visibility based on Smart Contract," International Journal of Advanced Computer Science and Applications(IJACSA), 11(10), 2020.

[29] Normaizeerah Mohd Noor, Noor Afiza Mat Razali, Nur Atiqah Malizan, Khairul Khalil Ishak, Muslihah Wook and Nor Asiakin Hasbullah, "Decentralized Access Control using Blockchain Technology for Application in Smart Farming" International Journal of Advanced Computer Science and Applications(IJACSA), 13(9), 2022.