UDC 004.4+004.9
DOI: 10.15587/1729-4061.2023.286079

DEVISING A METHOD FOR RAPID DATA RETRIEVAL USING EXPLORERS FOR BLOCKCHAIN ANALYSIS

Yaroslaw Dorogyy
Doctor of Technical Sciences, Associate Professor
Department of Information Systems and Technologies*
Vadym Kolisnichenko
Corresponding author
Postgraduate Student
Department of Computer Science
and Software Engineering*
E-mail: vadym.kolisnichenko@gmail.com
*National Technical University of Ukraine "Igor Sikorsky
Kyiv Polytechnic Institute"
Beresteiskyi (Peremohy) ave., 37, Kyiv, Ukraine, 03056

The object of this study is blockchain explorers and their usefulness in efficiently gathering data for blockchain network analysis. The process of blockchain analysis typically involves deploying and synchronizing a blockchain node, which requires significant computational resources and time for synchronization. Analyzing multiple blockchain networks simultaneously demands substantial effort and requires even greater costs.

The developed method involves utilizing publicly accessible blockchain explorers, which allows for rapid data retrieval with minimal computational resources for further analysis. Additionally, obtaining supplementary information from blockchain explorers provides valuable details that may be inaccessible using traditional data retrieval methods.

The efficiency of the proposed method was verified through the development of a prototype system. Data was collected for 14 specified blockchain networks to analyze smart contracts within these networks. Information about accounts (including balance statistics) was gathered, smart contracts were identified among the accounts, data on existing tokens owned by smart contracts was obtained, and bytecode and source code (where available) of contracts were collected and decompiled. The process took nearly 24 hours on a cloud computing machine with minimal configuration.

Based on the collected data, an example smart contract was analyzed to demonstrate the completeness of the process. The results of this research minimize computational resource expenses and allow for a simplified and rapid data gathering process without manual configuration, enabling researchers and analysts to concentrate on subsequent stages of analysis

Keywords: rapid data retrieval, blockchain analysis, blockchain explorers, multithreaded data processing

Received date 02.06.2023
Accepted date 18.08.2023
Published date 30.08.2023

***How to Cite**: Dorogyy, Y., Kolisnichenko, V. (2023). Devising a method for rapid data retrieval using explorers for blockchain analysis. Eastern-European Journal of Enterprise Technologies, 4 (2 (124)), 6–16. doi: https://doi.org/10.15587/1729-4061.2023.286079*

1. Introduction

Blockchain analysis is a fundamental and universal concept that serves as a key tool for understanding the intricacies of blockchain networks. Its scope encompasses a variety of methodologies and methods aimed at extracting valuable details and meaningful interpretations from these decentralized systems. Such analysis involves various aspects, including but not limited to transaction analysis, network analysis, technical analysis, etc.

Blockchain transaction analysis refers to the systematic study and interpretation of blockchain data in order to gain an in-depth understanding of transaction dynamics, recognize patterns, and identify anomalies. It is a powerful analytical tool that improves understanding of user behavior and important events in blockchain networks. As the fields of blockchain and decentralized finance (DeFi) develop rapidly, so does the potential for the use of transaction analysis.

The introduction of smart contracts revolutionized the way blockchain networks are used. Transactions that used to be limited to basic fund transfers now act as triggers to execute complex logic and transitions between different states. This advanced functionality includes a wide range of activities that go beyond financial transactions. Examples include actions in games [1] or data verification from higher-level blockchain networks. Smart contracts have ushered in a new era where transactions are enhanced, providing diverse and dynamic interactions in the blockchain ecosystem. The emergence of smart contracts significantly affected the field of analysis of blockchain transactions, leading to the emergence of a separate field – analysis of smart contracts. By reviewing smart contracts, researchers and auditors can gain insight into the behavior, security, and integrity of their work.

Blockchain analysis involves examining the data stored on the blockchain, which requires careful collection and organization for comprehensive evaluation. Analyzing blockchain networks usually involves setting up personal blockchain nodes and using the remote procedure call protocol encoded in JSON (JSON-RPC), which requires a powerful computing system. For full synchronization of nodes (Full Nodes), the recommended characteristics include a significant amount of RAM (random-access memory), a fast Internet connection, and a large volume of solid-state drive (SSD). The synchronization process can take days or even weeks. Analyzing multiple blockchain networks simultaneously requires considerable effort and expense.

In certain situations, there may be a need for operational analysis of several blockchain networks, especially when comparing their general characteristics. In such cases, it may be appropriate to use public nodes instead of setting up private ones. Public blockchain nodes typically provide a JSON-RPC interface for basic minimal interaction with the blockchain network. The interface makes it possible to retrieve a variety of information, such as transaction and account details, as well as send signed transactions, for example, to smart contracts

Therefore, scientific research into this area is important because it makes it possible to optimize and speed up the processes of data collection from blockchain networks.

The results of such studies are needed in practice because they minimize the cost of computing resources and allow for a simplified and accelerated data collection process, giving researchers the opportunity to focus on the next stages of analysis.

2. Literature review and problem statement

In work [2], a comprehensive review of applications, tasks, and methods of analysis of blockchain transactions was carried out. The authors identify three main tasks of transaction analysis: associating an address with an individual (establishing account owners), understanding the flow of transactions (processing massive sequences of transactions), and analyzing smart contracts (understanding the business logic of smart contracts). An integral part of any analysis is data collection, and blockchain transaction analysis is no exception. The work emphasizes the importance of cross-network blockchain analysis but does not indicate the means of implementing such analysis, including the part of data collection.

The analysis of blockchain transactions and smart contracts is mainly related to the study of data stored on the blockchain (on-chain data). These data require systematic collection and organization for comprehensive analysis. Once a blockchain node is deployed and synchronized, interfaces such as JSON-RPC can be used to retrieve the required data [3]. In cases where a more thorough and extensive analysis is required, direct communication with a synchronized database is preferred because of possible delays associated with the use of intermediate interfaces. However, for this study, the focus is on light but broad analysis scenarios that do not require a high level of intensity and depth of analysis.

An example of the analysis of blockchain transactions is given in [3]. The authors chose a blockchain network with a small amount of data – PIVX. The size of all data on this network was only 17 GB. The authors of the work upload data from the blockchain network to the database using the JSON-RPC interface and effectively analyze transactions in this network. But the question remains unsolved, how to scale this approach of transaction analysis for multiple networks.

Usually, blockchain analysis involves setting up a personal node. This approach requires the availability of sufficiently powerful computer systems. For Ethereum nodes, it is recommended [4] to have a minimum of 16 GB of RAM, a download speed of more than 25 Mbps, an SSD with a capacity of more than 650 GB for full node synchronization, and a 12 TB SSD for the archive node. Also, the sync process can take anywhere from a few days to a few weeks. Analyzing multiple blockchain networks simultaneously requires significant effort and comes with significant costs.

Techniques for accessing blockchain networks are described in [5]. The greatest attention is paid to three approaches: running a local full node, executing requests to the node as a service (node-as-service), and configuring a lightweight node. The work presents the minimum system requirements for different networks and compares different node-as-service providers (the number of protocols supported by the provider, minimum costs and other information). Although the described methods are effective for obtaining data from blockchain networks, they assume the deployment of a node for each individual network, which may be inefficient when analyzing several networks at once.

In article [6], the authors present an open-source system for blockchain analysis, with support for various networks. The system involves the use of a deployed full blockchain node from which it imports data. The developed system is quite effective in processing data uploaded by the blockchain node. In addition to communication with the node through the JSON-RPC interface, the system has a mode of direct parsing of the database without the use of intermediate network interfaces. In any case, the use of this system requires the deployment of a full blockchain node. In addition, the presented system only supports Bitcoin and derivative networks.

In [7], the authors develop a cluster-based system for parallel construction of a distributed graph of transactions for the application of various algorithms. The developed system allows efficient analysis of graphs for a large number of blockchain transactions. It is a sound approach for deep and complex analysis where intensive calculations are performed. The disadvantage of the presented method is the lack of emphasis on data collection and its application to a large number of blockchain networks. For a simpler but broad analysis, including a general analysis of several blockchain networks, this approach may be unreasonable and resource-consuming.

In [8], a system for analyzing blockchain transactions in the form of a blockchain explorer for the Bitcoin network is proposed. The system uses a number of methods for analysis, including statistical methods, known address clustering, pathfinding, and others. The limitation of the work is that it focuses only on one specific network and does not involve the analysis of several networks.

In [9], a method was developed to detect a financial pyramid based on the analysis of the byte code of a smart contract. To obtain the byte code of the smart contract, the authors use a blockchain explorer. Using a blockchain explorer is justified in this case, as it makes it possible to save resources and get the bytecode of the contract without deploying your own node. Although the described method of detecting a financial pyramid can theoretically be applied in many blockchain networks, the work does not indicate how to scale it in this network, let alone for other networks. Therefore, the limitation of the cited work is obtaining the necessary data. Blockchain explorer is used only as a method of obtaining information for a specific smart contract, not a mass analysis of contracts of many blockchain networks.

A general review of the above literature [2, 3, 5–9] allows us to state that scaling is an unsolved issue, and the main drawback of the described methods and solutions is their focus on a specific blockchain network (or derivatives of the same network), as well as, in most cases, the use of a deployed personal full node of a specific network. When conducting blockchain analysis, it is important to support several blockchain networks, since a typical situation is when a user converts the cryptocurrency of one network into the cryptocurrency of another and continues his activities in the last network. Scaling of the examined solutions is mostly resource-intensive and takes a significant part of the time in terms of setting up and synchronizing nodes. This allows us to state that it is appropriate to conduct a study aimed at devising another method for collecting information from blockchain networks, which could be fast and resource-efficient when analyzing several networks.

3. The aim and objectives of the study

The purpose of this study is to devise a method of rapid data collection from blockchain explorers for the analysis of blockchain networks. This will make it possible to obtain the necessary data simultaneously from several blockchain networks for further analysis quickly, without manual configuration of nodes and with minimal expenditure on computing resources. It will also improve existing systems for analyzing blockchain transactions.

To achieve the goal, the following tasks were set:

– to determine the main stages of the method of using explorers to rapidly obtain data for blockchain analysis;

– to analyze existing blockchain explorers and choose the most suitable one for use in further experiments;

– to propose the architecture of the system that will use the proposed method;

– to develop a prototype system for data collection and analysis of blockchain networks based on the designed architecture;

– to check the effectiveness of the proposed method of obtaining data for blockchain analysis.

4. The study materials and methods

The object of our research is blockchain explorers and their use in rapid data collection for the analysis of blockchain networks. The main hypothesis of the study is to quickly collect the necessary blockchain data using minimal computing resources.

14 blockchain networks based on the Ethereum virtual machine (EVM) were selected to be used for the information gathering experiment: Acala [10], Aves [11], Energy Web [12], Era Swap [13], Karura [14], Kava [15], MCHverse [16], Nahmii [17], Neatiio [18], OASYS [19], Puppynet [20], Rootstock [21], SmartBCH [22], Xiden [23] .

A cloud computing machine from the Vultr platform [24] with the least possible configuration for the selected operating system was used for the study. This is a normal performance machine with the following parameters:

– processor: 1 virtual processor (virtual CPU) Intel Core;

– RAM: 1 GB;

– drive: 25 TB SSD;

– outbound bandwidth: 1 TB;

– operating system: Ubuntu 23.04.

5. Results of investigating the method of obtaining data using blockchain explorers

5. 1. The main stages of the method of using explorers to rapidly obtain data for blockchain analysis

The method of using blockchain explorers to rapidly obtain data for analyzing blockchain networks includes the following actions.

Step 1. Choosing a blockchain explorer. The choice of the explorer is the first and main step and affects the effectiveness of the method. When choosing an explorer, it is important to take into account the task that the current blockchain analysis solves because this makes it clear what information will be collected and with what intensity, which will reveal potential limitations and feasibility of using the method. Among the criteria that can influence the choice of an explorer are:

– information provided by the explorer. Blockchain explorer is an intermediate interface between the blockchain network and the end user, so the data it provides to the end user is not completely identical to that stored on the blockchain but is pre-processed before being sent to the user. This is not necessarily a negative factor because during data processing, useful information is separated and efficiently aggregated, and can also be cached, which speeds up the overall process. In addition, blockchain explorers can provide additional information that is not stored on the blockchain, for example, source codes of verified smart contracts;

– availability of public explorer servers for selected networks in which the analysis is carried out;

– speed of communication with servers on which blockchain explorers are deployed and limitations on the number of requests;

– source code of the explorer. The open-source code of the blockchain explorer allows for a better understanding of its internal mechanisms, and as a result, allows for the identification of potentially inefficient parts and limitations. In addition, the source code of the explorer simplifies the software implementation of the proposed method. Also, it allows the integration of new blockchain networks that are not currently supported by the blockchain explorer.

Step 2. Architecture analysis. An analysis of the architecture of the existing or new blockchain network analysis system, into which the software implementation of the method will be integrated, is carried out. An important part of the architecture analysis is the determination of opportunities for scaling and parallelization of processes to simultaneously collect information from many blockchain networks.

Step 3. Determination of the method of communication with the blockchain explorer. It is determined exactly how information will be obtained from the blockchain explorer: using an application programming interface (API) or a regular web interface (which involves parsing web pages). The endpoints of the blockchain explorer are also defined, which will be used to obtain the necessary data.

Step 4. Software implementation of data collection. At this stage, software tools are analyzed and development of appropriate modules for communication with the blockchain explorer to obtain data based on the developed architecture taking into account parallelization is carried out.

5. 2. Choosing a blockchain explorer

Blockchain explorers are web tools designed to facilitate the exploration of blockchain networks. These tools allow users to efficiently search, filter, and sort transactions, blocks, accounts, and other information in the blockchain ecosystem. They have a wide range of applications, ranging from simple tasks such as verifying the existence of a token transfer transaction to more complex ones such as investigating certain criminal activities. Although the functionality of these tools is very similar to that provided by the JSON-RPC interface (mainly because explorers use this interface), they collect and provide additional information.

Owing to blockchain explorers, users get access to a number of functions that greatly simplify the process of interacting with blockchain data. For basic use cases, blockchain explorers allow users to quickly verify transactions, check balances, and track confirmations of specific transactions or blocks. More sophisticated explorers provide a list of smart contract methods and make it possible to interact with smart contracts. Moreover, some of the explorers provide information such as verified smart contracts [25], which means access to the source code associated with the deployed bytecode.

This study focuses on EVM compatible explorers. An open-source explorer is also preferred because it is easier for new blockchain networks to integrate them than to create their own explorers. Table 1 gives a comparison of different blockchain explorers according to the characteristics that are important for this study in order to choose the most suitable one.

Explorers usually have a similar interface. Fig. 1 shows the main page of the Blockscout explorer.

Table 1

Comparison of different blockchain explorer solutions

Explorer name	Open-source code	Supported networks	Smart contract bytecode	Verified source code of a smart contract
Blockchair [26]	No	17	No	No
Blockscout [27]	Yes	200+	Yes	Yes
Blockchain [28]	No	3	No	No
Bitquery [29]	Yes	50+	No	No
Etherscan [30]	No	19	Yes	Yes
Unmarshal Xscan [31]	No	18	No	No
Ethereum Beacon Chain [32]	Yes	4+	No	No
Otterscan [33]	Yes	–	No	Yes
Blockhead [34]	Yes	–	No	No
Ethernal [35]	Yes	–	No	No
3xpl [36]	No	20	No	No
EthVM [37]	Yes	2	No	No
BlockCypher [38]	Yes	7	No	No
BlockExplorer.one [39]	No	10	No	No
Tokenview [40]	No	120+	Yes	Yes
CoinMarketCap [41]	No	2	No	No

After analyzing various blockchain networks and explorers, it was concluded that Blockscout is the best candidate for further research purposes, although it may not be optimal for other tasks.

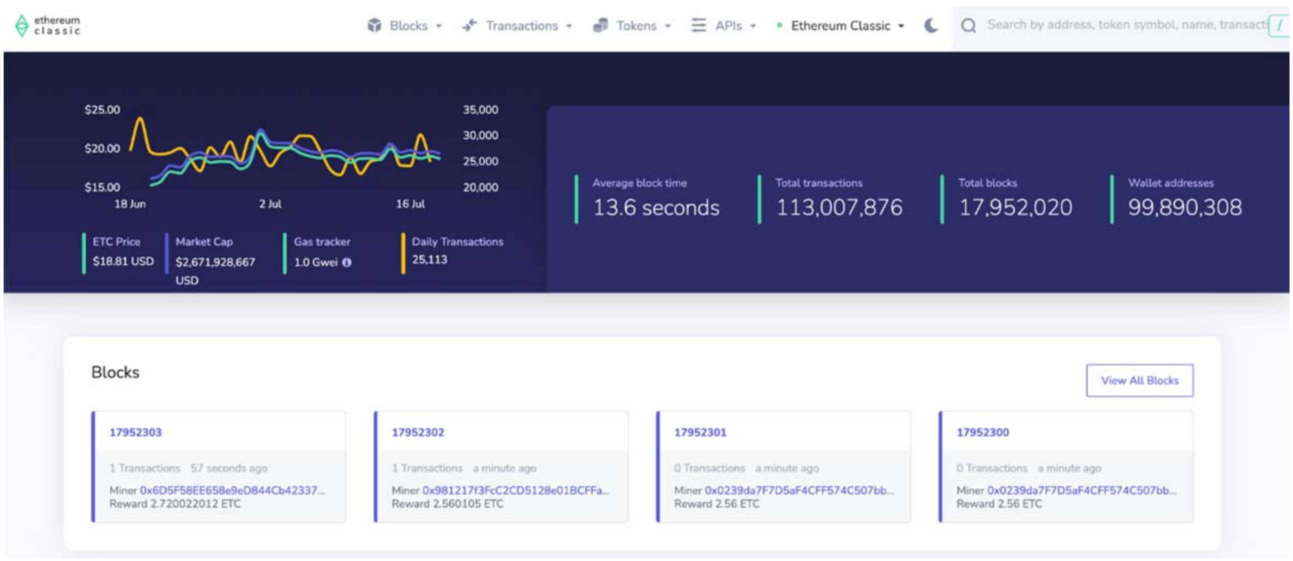
5. 3. System architecture for obtaining data for analyzing blockchain networks

This study focuses on the rapid acquisition of data simultaneously from many blockchain networks. The general structure of the system is quite simple and is shown in Fig. 2. It implies the sequence of data acquisition and payment functions.

The general approach works as follows. A list of blockchain networks (addresses of blockchain explorers) is provided at the input. The data is then retrieved from the explorers and processed. Based on the processed data and defined logic, additional information is obtained, which is also subject to processing. At the final stage, the obtained data and information are submitted for further analysis (manual or automated).

In this study, the scheme has a more complex structure (Fig. 3) and is focused on collecting information about existing smart contracts in blockchain networks. After each stage of acquisition, the response is processed to extract the necessary information and save it in a convenient format for offline access. First, the account information is obtained, then the smart contracts are filtered, the available tokens are obtained, the source code of the contracts is extracted if available, and the deployed bytecode is decompiled.

During the analysis phase, selected smart contract source codes are additionally pre-analyzed using the ChatGPT system [42] to provide a brief overview that may be useful during large-scale analysis. For this purpose, the API of the ChatGPT system is used [43]. Although other Large Language Models (LLM) can be used for this purpose, their detailed analysis, comparison, and selection is beyond the scope of this study. Here it serves to demonstrate a possible component of a large-scale analysis system.



ethereum classic
Blocks
Transactions
Tokens
APIs
Ethereum Classic
Search by address, token symbol, name, transaction
$25.00
$20.00
$15.00
35,000
30,000
25,000
20,000
18 Jun
2 Jul
16 Jul
ETC Price $18.81 USD
Market Cap $2,671,928,667 USD
Gas tracker 1.0 Gwei
Daily Transactions 25,113
Average block time 13.6 seconds
Total transactions 113,007,876
Total blocks 17,952,020
Wallet addresses 99,890,308
Blocks
View All Blocks
17952303
1 Transactions 57 seconds ago
Miner 0x6D5F58EE658e9eD844Cb42337...
Reward 2.720022012 ETC
17952302
1 Transactions a minute ago
Miner 0x981217f3Fc2CD5128e01BCFFa...
Reward 2.560105 ETC
17952301
0 Transactions a minute ago
Miner 0x0239da7F7D5aF4CFF574C507bb...
Reward 2.56 ETC
17952300
0 Transactions a minute ago
Miner 0x0239da7F7D5aF4CFF574C507bb...
Reward 2.56 ETC

Fig. 1. Main page of Blockscout explorer

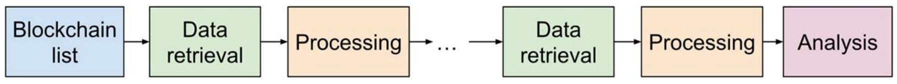


Fig. 2. General scheme of obtaining and processing data

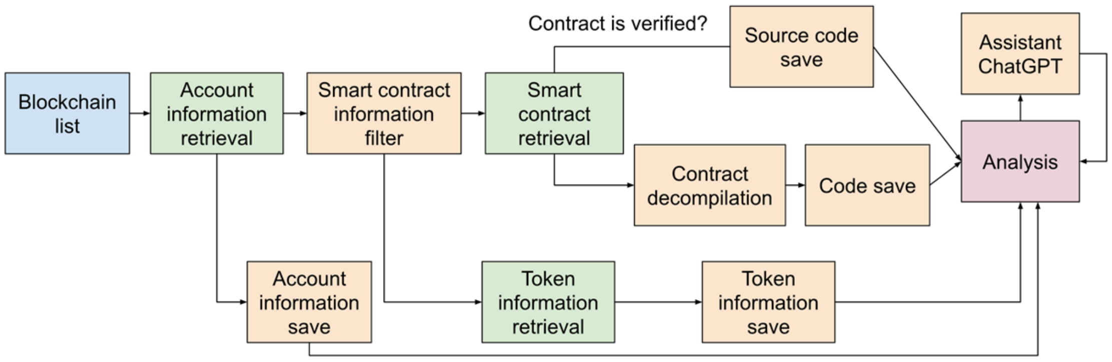


Fig. 3. Data collection and processing scheme for the proposed solution

This study uses the *gpt-3.5-turbo* model, which is the most efficient among the GPT-3.5 models. Models are non-deterministic, which means they produce different results for the same inputs. The value of the temperature parameter indicates the degree of randomness of the result, where 0 gives the most deterministic result, and 2 – the most random. A value of 0.3 is used in this study. As for the request sent to ChatGPT, it has the following structure: "Give a brief summary of the next Smart Contract:" and the source code of the smart contract is attached.

5. 4. Development of a prototype system for data collection and analysis of blockchain networks

The system prototype is built on the basis of the architecture developed in the previous subchapter. The architecture provides for the simultaneous acquisition of data from several blockchain networks for the analysis of smart contracts.

The Python programming language was used for the prototype, as well as the Beautiful Soup library [44] for parsing HyperText Markup Language (HTML). These tools are effective for prototyping and hypothesis testing.

As part of this data collection process, it is clear that there may be situations where the source code of contracts may not be available. In such cases, a method known as bytecode decompilation is used to produce pseudocode.

Bytecode decompilation is the process of reverse engineering the executable bytecode that represents the low-level instructions of a smart contract. By applying specialized tools and algorithms, the bytecode is transformed back into a more human-readable form that resembles the structure and logic of the original source code. This makes it possible to get details about the core functionality, implementation details, and possible vulnerabilities of a smart contract, despite not having the original source code.

The decompiler Panoramix decompiler [45] was selected because it is actively supported, open-source, stable, and produces acceptable results. Panoramix converts EVM bytecode into Python pseudocode.

In order to get data from Blockscout, the links that are responsible for providing certain data should be determined. Only three Blockscout explorer endpoints are used in this study:

– */accounts* – to get account addresses, cryptocurrency balances and account types, be it smart contract or Externally Owned Account (EOA). This endpoint provides only top accounts whose cryptocurrency balance exceeds 0. It is worth noting that native contracts (native smart contracts – integrated into the blockchain system, and not deployed in it) can be recognized as EOA and not as smart contracts;

– */address/{address}/contracts* – to get the smart contract source code (if available) and the deployed bytecode. Also, if the contract is a proxy contract [46], Blockscout detects it and provides the implementation address;

– */address/{address}/token-balances* – to get the tokens owned by the account and their quantity.

Data retrieval, parsing, and processing can be parallelized using a thread pool or process pool [47]. A simplified process of parallel data acquisition is shown in Fig. 4.

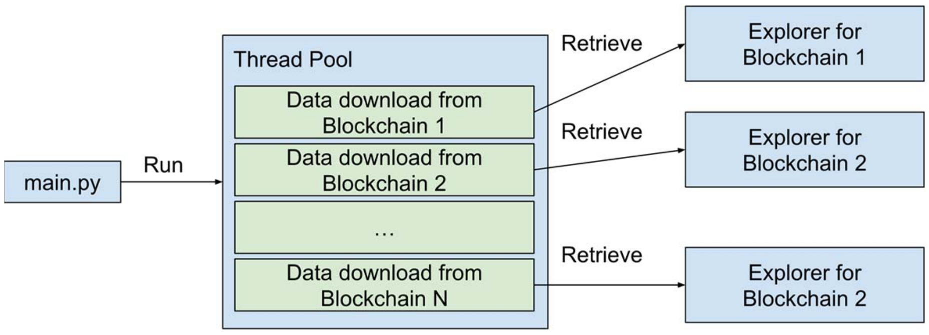

main.py
Run
Thread Pool
Data download from Blockchain 1
Data download from Blockchain 2
...
Data download from Blockchain N
Retrieve
Retrieve
Retrieve
Explorer for Blockchain 1
Explorer for Blockchain 2
Explorer for Blockchain 2

Fig. 4. Parallel acquisition of data using a thread pool

This will avoid various bottlenecks, among which the main ones are network latency and smart contract bytecode decompilation.

5. 5. Checking the effectiveness of the proposed method

During the first stage, information about accounts and cryptocurrency balance for each individual account in the network was obtained from blockchain explorers (via the*/accounts* endpoint). Table 2 displays the total number of accounts with a positive cryptocurrency balance received on the network, as well as account balance statistics. The total balance of all accounts is calculated as the sum of the balances of each network account. The maximum balance value is the largest balance among all network accounts. The average value of the balance is the arithmetic mean, that is, the ratio of the total balance to the number of received network accounts. Median balances are the value located in the middle of the sorted series of balances of all received network accounts. Balances are represented in the currency of the network and cannot be directly compared without prior conversion.

After receiving account data, smart contracts are selected for further processing. Table 3 displays the same information as the previous table but only for a subset of accounts – for smart contracts. The values are calculated in the same way as for Table 2 but, in this case, the values are taken only for smart contracts.

The next step involves getting the bytecodes of the deployed smart contracts from the blockchain explorers (via the */address/{address}/contracts* endpoint), as well as their source code, if available. All bytecodes are then decompiled using the selected decompiler. Table 4 shows the number of deployed bytecodes in each network, their average size, and the number of received source codes. It also indicates how many bytecodes were successfully decompiled.

The average size of a smart contract is calculated as the ratio of the sum of the sizes of all received bytecodes to their number received in a given blockchain network. Source codes are available only for those smart contracts that have been submitted to the blockchain explorer for verification. All received bytecodes are subject to the decompilation process, even when the corresponding source code is available. Successfully decompiled smart contract bytecode refers to the successful execution of the decompiler program and does not take into account the quality of the decompilation result.

At the next stage, information is obtained from the explorer (via the endpoint */address/{address}/token-balances*) about tokens that belong to smart contracts.

Table 2

Account information (including smart contracts) and cryptocurrency balance statistics

Network	Number of accounts	Balance of all accounts			
		Total	Maximum	Average	Median
Acala	1224	171484729.48	168607898.02	140101.90	28.82
Aves	2352	6673004.30	1760963.98	2837.16	0.06
Energy Web	50110	74200897.13	12580691.64	1480.76	0.19
Era Swap	16121	9100000000.00	5433176313.50	564481.11	10.35
Karura	5858	1164149.56	378459.08	198.73	0.29
Kava	1581668	120557035.14	54867879.90	76.22	0.00
MCH-verse	942	447838.61	201359.23	475.41	1.45
Nahmii	698	285.31	63.60	0.41	0.01
Neatiio	787	2679038.77	1077002.95	3404.12	100.00
OASYS	24140	10012023660.41	3557500019.90	414748.29	1.00
Puppynet	2362175	249907117.84	220266275.52	105.80	0.31
Rootstock	70371	21000141.98	20996519.35	298.42	0.00
SmartBCH	32289	20999852.15	20931686.58	650.37	0.00
Xiden	7218	300006996.32	299864128.00	41563.73	0.05

Table 3

Smart contract information and cryptocurrency balance statistics

Network	Number of accounts	Smart contract balance			
		Total	Maximum	Average	Median
Acala	5	723.74	705.09	144.75	2.00
Aves	20	1639250.45	1638909.31	81962.52	3.99
Energy Web	159	29609148.74	9420942.57	186221.06	0.11
Era Swap	1514	7429415264.35	5433176313.50	4907143.50	26221.50
Karura	3	4560.32	4417.80	1520.11	130.01
Kava	63	1988393.73	1915785.34	31561.81	1.09
MCH-verse	4	276566.01	201359.23	69141.50	37603.39
Nahmii	604	278.09	63.60	0.46	0.01
Neatiio	1	1.00	1.00	1.00	1.00
OASYS	20	939600812.54	912738642.13	46980040.63	111558.39
Puppynet	5	8117.42	7668.42	1623.48	200.00
Rootstock	528	2849.93	1603.85	5.40	0.00
SmartBCH	152	16215.19	15171.61	106.68	0.01
Xiden	6	13169.73	12230.20	2194.96	212.07

Table 4

Smart contract information

Network	Average contract size (bytes)	Received contracts	Resulting source codes	Decompiled successfully
Acala	5398.20	5	1	4
Aves	5495.06	17	2	14
Energy Web	3940.60	159	26	123
Era Swap	353.15	1509	1457	1255
Karura	940.33	3	1	2
Kava	9403.30	63	20	57
MCH-verse	5425.25	4	0	4
Nahmii	1854.62	604	0	513
Neatiio	6649.00	1	0	1
OASYS	1817.90	20	6	17
Puppynet	8159.78	5	0	5
Rootstock	1193.87	528	23	427
SmartBCH	10246.34	152	10	127
Xiden	2534.00	6	0	4

To demonstrate the effectiveness of the proposed approach, the time of data acquisition for further analysis was measured. The data acquisition time for each of the above-described stages and the total time are given in Table 5.

Table 5

Data acquisition time

Network	Loading time (sec)			Total (sec)
	Accounts	Contracts	Tokens	
Acala	58.62	12.99	6.69	78.30
Aves	67.53	19.68	12.45	99.74
Energy Web	889.63	92.88	53.21	1035.94
Era Swap	687.97	7559.63	1453.06	9700.73
Karura	252.03	5.97	3.50	261.56
Kava	48526.23	63.46	34.23	48628.76
MCH-verse	52.05	10.95	5.86	68.85
Nahmii	18.48	426.28	295.44	740.32
Neatiio	2100.09	131.61	130.85	2362.55
OASYS	644.77	23.35	14.11	682.37
Puppynet	86831.80	23.28	3.91	86869.73
Rootstock	1254.55	563.12	166.25	1984.23
SmartBCH	868.84	134.77	96.04	1099.85
Xiden	349.87	6.48	4.26	360.64

After receiving and processing all data, the analysis process was carried out. To demonstrate the process, a random smart contract was chosen from the set of verified contract source codes obtained – a contract from the Aves network with the address *0x75fA5fecE2A9783e28856c1A-7EA3Af544690ebc8* [48]. Fig. 5 demonstrates a part of the obtained source code [49]. Fig. 6 shows its corresponding decompiled bytecode version [50].

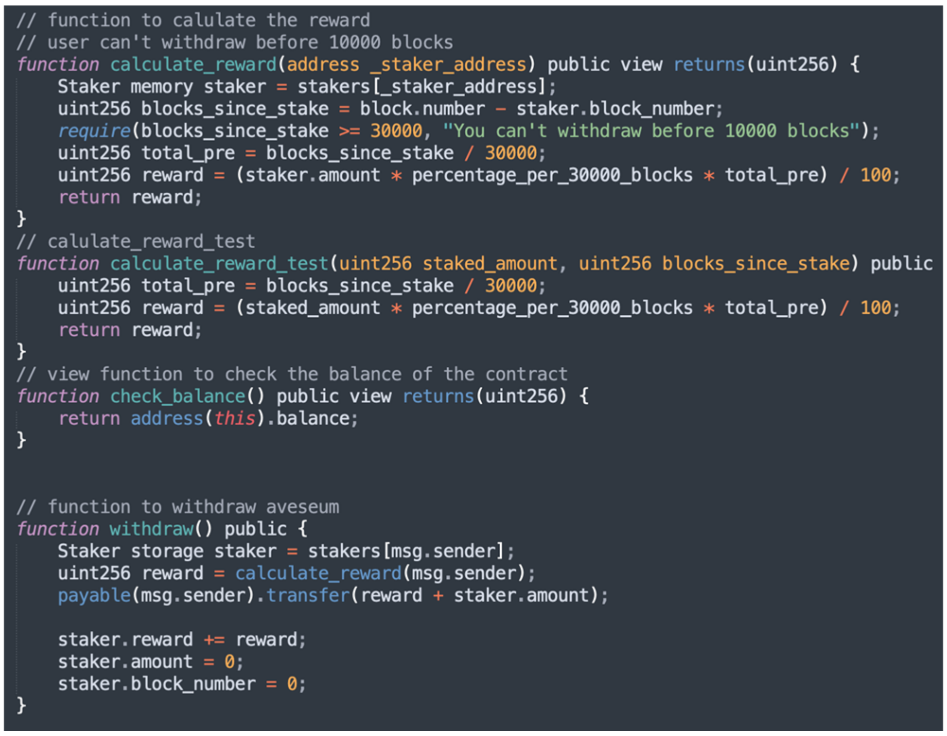
Before conducting a manual analysis of the contract source code, the ChatGPT assistant was asked to provide a brief overview. The answer is given below:

«The next smart contract is called StakeAVS. It allows users to stake aves tokens and receive rewards based on the amount staked and the duration of the stake. The contract is initialized with an owner address.

The contract includes a struct called Staker, which stores information about each staker, including the staked amount, the block number at which the stake was made, the staker's address, and the reward earned.

The contract has several functions. The calculate_reward function calculates the reward for a staker based on the number of blocks since the stake was made. The check_balance function allows users to check the balance of the contract. The withdraw function allows stakers to withdraw their staked aves tokens and their earned rewards. The premature_withdraw function allows stakers to withdraw their staked aves tokens before the required number of blocks have passed, but they will not receive any rewards. The stake function allows users to stake aves tokens. The check_reward function allows users to check their earned rewards. The check_staked function allows users to check the amount of aves tokens they have staked. The check_block_number function allows users to check the block number at which they made their stake. The change_owner function allows the owner to change the owner address. The withdraw_aves function allows the owner to withdraw aves tokens from the contract. The stop_staking function allows the owner to stop staking, while the start_staking function allows the owner to resume staking.

The contract also includes a receive function, which allows the contract to receive aves tokens as funds from stakers.»

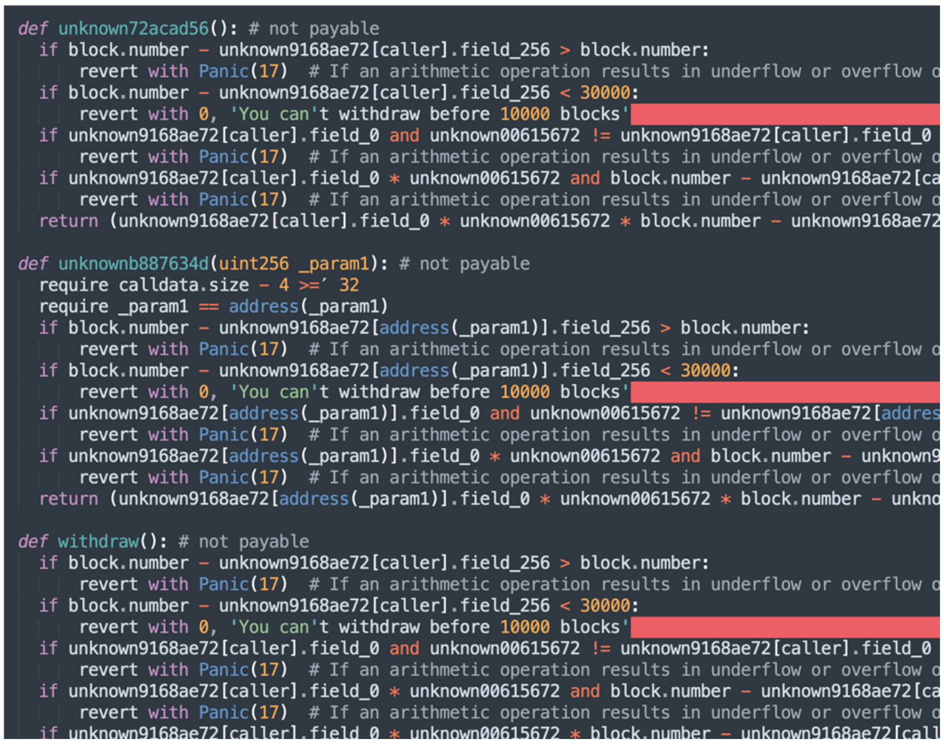


```
// function to calulate the reward
// user can't withdraw before 10000 blocks
function calculate_reward(address _staker_address) public view returns(uint256) {
    Staker memory staker = stakers[_staker_address];
    uint256 blocks_since_stake = block.number - staker.block_number;
    require(blocks_since_stake >= 30000, "You can't withdraw before 10000 blocks");
    uint256 total_pre = blocks_since_stake / 30000;
    uint256 reward = (staker.amount * percentage_per_30000_blocks * total_pre) / 100;
    return reward;
}
// calulate_reward_test
function calculate_reward_test(uint256 staked_amount, uint256 blocks_since_stake) public
    uint256 total_pre = blocks_since_stake / 30000;
    uint256 reward = (staked_amount * percentage_per_30000_blocks * total_pre) / 100;
    return reward;
}
// view function to check the balance of the contract
function check_balance() public view returns(uint256) {
    return address(this).balance;
}


// function to withdraw aveseum
function withdraw() public {
    Staker storage staker = stakers[msg.sender];
    uint256 reward = calculate_reward(msg.sender);
    payable(msg.sender).transfer(reward + staker.amount);

    staker.reward += reward;
    staker.amount = 0;
    staker.block_number = 0;
}
```

Fig. 5. The source code of the received smart contract



```
def unknown72acad56(): # not payable
  if block.number - unknown9168ae72[caller].field_256 > block.number:
      revert with Panic(17)  # If an arithmetic operation results in underflow or overflow o
  if block.number - unknown9168ae72[caller].field_256 < 30000:
      revert with 0, 'You can't withdraw before 10000 blocks'
  if unknown9168ae72[caller].field_0 and unknown00615672 != unknown9168ae72[caller].field_0
      revert with Panic(17)  # If an arithmetic operation results in underflow or overflow o
  if unknown9168ae72[caller].field_0 * unknown00615672 and block.number - unknown9168ae72[ca
      revert with Panic(17)  # If an arithmetic operation results in underflow or overflow o
  return (unknown9168ae72[caller].field_0 * unknown00615672 * block.number - unknown9168ae72

def unknownb887634d(uint256 _param1): # not payable
  require calldata.size - 4 >=′ 32
  require _param1 == address(_param1)
  if block.number - unknown9168ae72[address(_param1)].field_256 > block.number:
      revert with Panic(17)  # If an arithmetic operation results in underflow or overflow o
  if block.number - unknown9168ae72[address(_param1)].field_256 < 30000:
      revert with 0, 'You can't withdraw before 10000 blocks'
  if unknown9168ae72[address(_param1)].field_0 and unknown00615672 != unknown9168ae72[addres
      revert with Panic(17)  # If an arithmetic operation results in underflow or overflow o
  if unknown9168ae72[address(_param1)].field_0 * unknown00615672 and block.number - unknown9
      revert with Panic(17)  # If an arithmetic operation results in underflow or overflow o
  return (unknown9168ae72[address(_param1)].field_0 * unknown00615672 * block.number - unkno

def withdraw(): # not payable
  if block.number - unknown9168ae72[caller].field_256 > block.number:
      revert with Panic(17)  # If an arithmetic operation results in underflow or overflow o
  if block.number - unknown9168ae72[caller].field_256 < 30000:
      revert with 0, 'You can't withdraw before 10000 blocks'
  if unknown9168ae72[caller].field_0 and unknown00615672 != unknown9168ae72[caller].field_0
      revert with Panic(17)  # If an arithmetic operation results in underflow or overflow o
  if unknown9168ae72[caller].field_0 * unknown00615672 and block.number - unknown9168ae72[ca
      revert with Panic(17)  # If an arithmetic operation results in underflow or overflow o
  if unknown9168ae72[caller].field_0 * unknown00615672 * block.number - unknown9168ae72[call
```

Fig. 6. Decompiled code of the received smart contract

The next stage involves manual analysis. The *calculate_reward* function provides a reward of 1 % of the stake for every 30,000 blocks. According to information from explorer [51], the current average block time is 10.3 seconds, which means that a 1 % reward is given every 309,000 seconds. This means that within a year (31536000 seconds), the bet owner will receive almost 100 % profit from his investment.

Another important aspect of this contract is the *withdraw_aves* function, which allows the contract owner to withdraw all funds at any time, thereby creating a situation where all depositors lose their investment. In addition, if the balance of the contract is formed only from investments, then it is not known where the contract takes the funds to cover the rewards. These details make the contract very risky to use.

6. Discussion of results of research on the method of obtaining data using blockchain explorers

The method of using blockchain explorers to rapidly obtain data from many blockchain networks includes 4 stages: choosing a blockchain explorer; analysis of the system architecture into which the software implementation of data collection will be integrated; determining the method of communication with the blockchain explorer; software implementation of data collection from blockchain explorers.

For data collection, it is possible to use ordinary public nodes and JSON-RPC interface but blockchain explorers provide significant advantages over a simple JSON-RPC interface. They provide additional information that contributes to a more complete analysis, especially for understanding smart contracts in the blockchain networks under investigation. Tables 2, 3 give general statistical information on received top-accounts and their balances, which can be used, for example, to determine priorities in further manual or automated analysis. Table 4 presents the number of received source codes of smart contracts for different types of networks. The source codes of smart contracts are the information that cannot be obtained using the usual JSON-RPC interface of the blockchain node since the nodes do not store the source codes. In addition, Table 4 gives information about deployed smart contracts.

Some explorers are designed specifically for certain networks [52], while others support different types of networks [53]. Although explorers have different characteristics, functionality, and limitations [54], in most cases they can be used interchangeably. Among possible explorers, Blockscout was chosen for this study. Blockscout is a platform designed for in-depth research into blockchain systems. Multi-blockchain support, extensive customization, and open source provide users with full access to all data and functions for various blockchain networks. Thus, Blockscout provides users with a wide range of opportunities to deeply explore and understand the blockchain space. One important feature is that Blockscout provides verified contract source codes and its corresponding deployed bytecode to be used in further analysis.

Although Blockscout provides an API for use by third-party applications [55], in many cases it is disabled or restricted for public use. Because of these limitations, it is reasonable to perform web scraping and parsing [56]. Fig. 1 shows the web interface, the main page of Blockscout Explorer. It is the web interface that is subject to web scraping and parsing. There are two main potential problems with this approach. First, web scraping depends on the specific structure and layout of the website. Any changes or updates to the website design may interfere with the scraping process, requiring constant maintenance and updating of the scraping code. A second potential problem could be imposed speed limits or blocked IP addresses when excessive scraping activity is detected on web servers. This can disrupt the data collection process and possibly lead to problems accessing the server.

The developed architecture of the system (Fig. 3) is aimed at obtaining data and analyzing blockchain networks, namely, smart contracts. A list of networks is given to the system input, and all the necessary information for smart contract analysis is given to the analysis phase, including a summary of the smart contract generated by the ChatGPT assistant. This architecture is just an example of what a system might look like. In fact, it can be much more complex and include even more components for a more complete analysis but the construction of such a system is beyond the scope of this study.

Based on this architecture, a multi-threaded (Fig. 4) system prototype was designed for further experiments. Although decompilation and the ChatGPT assistant are not part of information retrieval step, they are part of information processing. Therefore, to demonstrate a full-fledged example of blockchain analysis, we included these components in the current work.

As regards decompilation, online solutions such as Dedaub's EVM Bytecode Decompiler [57] and Online Solidity Decompiler [51] provide the best results. But for this analysis, the offline version was chosen since the priority is the speed of obtaining results, and not the readability or accuracy of the resulting pseudocode. However, even poorly decompiled bytecode can be improved. To preserve overall processing speed and obtain improved decompilation results, online versions of decompilers can be used only for selected contracts.

The multi-threaded implementation (Fig. 4) made it possible to collect data from several blockchain systems at the same time. Table 5 gives the total time of information collection for each investigated network, and the time spent at each stage of collection: obtaining information about accounts, obtaining smart contracts, obtaining information about tokens owned by the contract. The amount of time spent collecting information depends on the amount of information in a specific blockchain network and the limitations of the server from which the data is received. Due to the multi-threaded implementation, the total time of the entire data acquisition process is the largest total data acquisition time from a single network. In this experiment, it took the most time (Table 5) to receive data from the Puppynet network – 86869.73 seconds, which is almost 24 hours. Therefore, the whole process lasted up to 24 hours.

Using minimal resources, it became possible to collect the necessary information, available smart contract source codes, and deployed bytecodes from several blockchain networks for further analysis. After that, as an example, an analysis of the contract was carried out based on the collected data and the source code of the contract (Fig. 5). The analysis of the smart contract provided information about its behavior and role, as well as the risks of its use.

It can be argued that the use of blockchain explorers is a justified method for obtaining data and useful information in a short period of time using a small amount of resources. To be precise, there is no need to rent powerful servers and manually configure nodes for each blockchain network and synchronize them for days or weeks.

The proposed method has drawbacks. If the server on which the explorer is deployed is slow or artificially limits the speed, you will have to spend more time collecting the necessary data. Sometimes the blockchain node used by the explorer may not be fully synchronized and, as a result, may provide out-of-date data. In addition, not all blockchain networks may have a publicly available instance of the chosen explorer.

If some network in the research area has the listed problems, it may be reasonable to deploy a local blockchain node and explorer. This will allow using the same approach and avoid different implementations.

The main limitation of this approach is large blockchain networks, as they involve an extremely large amount of data. In this case, the blockchain explorer will be another intermediate interface that introduces delay. Therefore, it is rational to choose another data retrieval technique or to download only a previously limited set of data.

The development of this research may consist in expanded types of public data sources, including other types of blockchain explorers. This will make it possible to support an even larger number of blockchain networks, receive data faster, and have backup sources, in case of restrictions or disconnection of the main ones. In addition, another area of development of this research may be its application, namely, in continuous scanning and data retrieval. This will make it possible to observe the defined parameters of many blockchain networks in real time and, in case of any, to receive notifications about these changes. In the further development of this research, possible technical difficulties are associated with the limitations of information sources and the servers on which they are located.

7. Conclusions

1. The main stages of the method of using blockchain explorers for rapid simultaneous retrieval of data from many blockchain networks for blockchain analysis have been defined, namely:

– choosing a blockchain explorer;

– analysis of the architecture of the existing or new blockchain network analysis system, into which the software implementation of the method will be integrated;

– determination of the communication technique with the blockchain explorer;

– software implementation of data acquisition from blockchain explorers.

2. Existing blockchain explorers were analyzed and Blockscout was selected as the optimal one for use in this study. This explorer is available for more than 200 blockchain networks, is open-source, and can provide data about a deployed EVM smart contract, including its bytecode, source code (if available), and proxy contract details.

3. The system architecture is proposed for obtaining data from blockchain explorers about deployed smart contracts in the blockchain network, and for their further analysis. The system includes a series of blocks for data acquisition, processing, and storage. The architecture provides for the use of ChatGPT as an assistant for the analysis of smart contracts.

4. Based on the given architecture, a multi-threaded system prototype has been developed that makes it possible to receive data simultaneously from many blockchain networks using publicly available instances of blockchain explorers. The system collects data about smart contracts for their further analysis. The bytecodes of the deployed smart contracts are decompiled in the process.

5. The effectiveness of the proposed method was tested using the developed system prototype. For the given 14 blockchain networks, data was collected for the analysis of smart contracts of the networks. Account data was collected (including balance statistics), smart contracts were selected among the accounts, data was obtained on existing tokens (owned by

smart contracts), bytecodes of contracts and their source codes (where available) were collected, and their decompilation was carried out. The process took almost 24 hours and cost up to USD 1 for a selected cloud computing machine with minimal configuration. Based on the collected data, a random smart contract was analyzed to illustrate the completeness of the process.

Conflicts of interest

The authors declare that they have no conflicts of interest in relation to the current study, including financial, personal, authorship, or any other, that could affect the study and the results reported in this paper.

Funding

The study was conducted without financial support.

Data availability

The manuscript has associated data in the data warehouse.

References

1. Iyer, K., Dannen, C. (2018). Building Games with Ethereum Smart Contracts. Apress Berkeley, 269. doi: https://doi.org/10.1007/978-1-4842-3492-1
2. Dorogyy, Y., Kolisnichenko, V. (2023). Blockchain Transaction Analysis: A Comprehensive Review of Applications, Tasks and Methods. System research and information technologies. (In Press)
3. Werner, R., Lawrenz, S., Rausch, A. (2020). Blockchain Analysis Tool of a Cryptocurrency. Proceedings of the 2020 The 2nd International Conference on Blockchain Technology. doi: https://doi.org/10.1145/3390566.3391671
4. Hardware requirements. Go-Ethereum. URL: https://geth.ethereum.org/docs/getting-started/hardware-requirements
5. Luo, Z., Murukutla, R., Kate, A. (2022). Last Mile of Blockchains: RPC and Node-as-a-service. arXiv. doi: https://doi.org/10.48550/arXiv.2212.03383
6. Kalodner, H., Möser, M., Lee, K., Goldfeder, S., Plattner, M., Chator, A., Narayanan, A. (2020). BlockSci: Design and applications of a blockchain analysis platform. 29th USENIX Security Symposium, 2721–2738. URL: https://www.usenix.org/system/files/sec20-kalodner.pdf
7. Kılıç, B., Özturan, C.Sen, A. (2022). Parallel analysis of Ethereum blockchain transaction data using cluster computing. Cluster Computing, 25 (3), 1885–1898. doi: https://doi.org/10.1007/s10586-021-03511-0
8. Kuzuno, H., Karam, C. (2017). Blockchain explorer: An analytical process and investigation environment for bitcoin. 2017 APWG Symposium on Electronic Crime Research (ECrime). doi: https://doi.org/10.1109/ecrime.2017.7945049
9. Wen, X., Yeo, K. S., Wang, Y., Cheng, L., Zhu, F., Zhu, M. (2023). Code Will Tell: Visual Identification of Ponzi Schemes on Ethereum. Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems. doi: https://doi.org/10.1145/3544549.3585861
10. Acala chain explorer. Acala.network. URL: https://blockscout.acala.network/
11. Aves explorer. Avescan.io. URL: https://avescan.io
12. Energy web chain energy web foundation explorer. Energyweb.org. URL: https://explorer.energyweb.org
13. Eraswap explorer. Eraswap.Info. URL: https://eraswap.info
14. Karura chain explorer. Karura.network. URL: https://blockscout.karura.network
15. Kava Ethereum Co-Chain Explorer. Kava.io. URL: https://explorer.kava.io
16. MCH verse explorer. Mycryptoheroes.net. URL: https://explorer.oasys.mycryptoheroes.net
17. Nahmii explorer. Nahmii.io. URL: https://explorer.nahmii.io
18. Neatio. Neatio.net. URL: https://scan.neatio.net
19. Oasys explorer. Oasys.Games. URL: https://scan.oasys.games
20. BONE BONE explorer. Shib.io. URL: https://puppyscan.shib.io
21. Rootstock (RBTC) explorer. Blockscout.com. URL: https://blockscout.com/rsk/mainnet
22. SmartBCH explorer. Sonar.Cash. URL: https://sonar.cash
23. Xiden explorer. Xiden.com. URL: https://explorer.xiden.com
24. SSD VPS Servers, Cloud Servers and Cloud Hosting. Vultr.com. URL: https://www.vultr.com/
25. Ethereum. Verified Contracts. Etherscan.io. URL: https://etherscan.io/contractsVerified/
26. Blockchair - Universal blockchain explorer and search engine. Blockchair.com. URL: https://blockchair.com/
27. Chains & projects using blockscout. Blockscout.com. URL: https://docs.blockscout.com/about/projects
28. Blockchain Explorer - Bitcoin Tracker & More. Blockchain.com. URL: https://www.blockchain.com/explorer
29. Blockchain Explorer By Bitquery. Bitquery Explorer. URL: https://explorer.bitquery.io/
30. Etherscan Explorer Services. Etherscan.io. URL: https://etherscan.io/eaas
31. Unmarshal Blockchain Explorer. Xscan.io. URL: https://xscan.io/
32. Open source Ethereum blockchain explorer. Beaconcha.In. URL: https://beaconcha.in/
33. otterscan: A blazingly fast, local, Ethereum block explorer built on top of Erigon. URL: https://github.com/otterscan/otterscan
34. Blockhead - track, visualize & explore all of crypto, DeFi & web3. Blockhead.Info. URL: https://blockhead.info/explorer

35. ethernal: Ethernal is a block explorer for EVM-based chains. URL: https://github.com/tryethernal/ethernal
36. 3xpl. URL: https://3xpl.com/
37. ethVM: An Open Source Block Explorer for Ethereum with Users In Mind. URL: https://github.com/EthVM/EthVM
38. explorer: Block explorer showcasing the BlockCypher APIs. URL: https://github.com/blockcypher/explorer
39. Search for block, transaction, address. Blockexplorer.One. URL: https://blockexplorer.one/
40. The General Multi-chain Explorer and Blockchain API. Tokenview.io. URL: https://tokenview.io/
41. Blockchain Explorer. Coinmarketcap.com. URL: https://blockchain.coinmarketcap.com/
42. OpenAI (2023). GPT-4 Technical Report. arXiv. doi: https://doi.org/10.48550/arXiv.2303.08774
43. OpenAI platform. Openai.com. URL: https://platform.openai.com/docs/api-reference
44. Beautifulsoup4. PyPI. URL: https://pypi.org/project/beautifulsoup4/
45. palkeo. panoramix: Ethereum decompiler. URL: https://github.com/palkeo/panoramix
46. Meisami, S., Bodell, W. E. (2023). A Comprehensive Survey of Upgradeable Smart Contract Patterns. arXiv. doi: https://doi.org/10.48550/arXiv.2304.03405
47. multiprocessing - Process-based parallelism. Python Documentation. URL: https://docs.python.org/3/library/multiprocessing.html
48. StakeAVS (0x75fA5fecE2A9783e28856c1A7EA3Af544690ebc8) - explorer. Avescan.Io. URL: https://avescan.io/address/0x75fA5fecE2A9783e28856c1A7EA3Af544690ebc8
49. Stake.Sol. URL: https://gist.github.com/VaWheel/072250e3b419fb9ad8e5c9b411776579
50. StakeDecompiled.Py. URL: https://gist.github.com/VaWheel/ad80df2f8b15067388876e92d8b80901
51. Online Solidity Decompiler. URL: https://ethervm.io/decompile
52. RSK explorer. Rsk.Co. URL: https://explorer.rsk.co/
53. Blockchair - Crunchbase Company Profile & Funding. Crunchbase.com. URL: https://www.crunchbase.com/organization/blockchair
54. Block Explorers. Alchemy.com. URL: https://www.alchemy.com/best/block-explorers
55. API - Blockscout. Blockscout.com. URL: https://docs.blockscout.com/for-users/api
56. Brenning, A., Henn, S. (2023). Web scraping: a promising tool for geographic data acquisition. arXiv. doi: https://doi.org/10.48550/arXiv.2305.19893
57. EVM Bytecode Decompiler. Dedaub.com. URL: https://library.dedaub.com/decompile