PonziLens+: Visualizing Bytecode Actions for Smart Ponzi Scheme Identification

Xiaolin Wen, Tai D. Nguyen, Shaolun Ruan, Qiaomu Shen, Jun Sun, Feida Zhu, and Yong Wang

***Abstract*—With the prevalence of smart contracts, smart Ponzi schemes have become a common fraud on blockchain and have caused significant financial loss to cryptocurrency investors in the past few years. Despite the critical importance of detecting smart Ponzi schemes, a reliable and transparent identification approach adaptive to various smart Ponzi schemes is still missing. To fill the research gap, we first extract semantic-meaningful actions to represent the execution behaviors specified in smart contract bytecodes, which are derived from a literature review and in-depth interviews with domain experts. We then propose *PonziLens+*, a novel visual analytic approach that provides an intuitive and reliable analysis of Ponzi-scheme-related features within these execution behaviors. *PonziLens+* has three visualization modules that intuitively reveal all potential behaviors of a smart contract, highlighting fraudulent features across three levels of detail. It can help smart contract investors and auditors achieve confident identification of any smart Ponzi schemes. We conducted two case studies and in-depth user interviews with 12 domain experts and common investors to evaluate *PonziLens+*. The results demonstrate the effectiveness and usability of *PonziLens+* in achieving an effective identification of smart Ponzi schemes.**

***Index Terms*—Smart Ponzi Scheme, Visual Analytics, Blockchain, Smart Contracts.**

I. Introduction

PONZI scheme [1] is a classic financial fraud that appeared in the offline world over 150 years ago. It uses the investment of new investors to compensate the existing investors, and will inevitably collapse and cause financial losses to most investors. With the rapid development of blockchain technology, Ponzi schemes have also become widely spread on blockchain platforms (e.g., Ethereum[1]) in the past few years [2], [3]. Such Ponzi schemes leverage smart contracts on the blockchain and are run in a decentralized, anonymous, and immutable manner, which are called **smart Ponzi schemes** [4]. Smart Ponzi schemes often lure investors by posing as high-profit investment plans [4]. For example, Forsage, a platform marketed as a low-risk investment using smart contracts, was revealed to be a $340 million global Ponzi scheme [5]. At present, smart Ponzi schemes have become one of the most common frauds for cryptocurrency [6], [7]. Ponzitracker [8] reported that the total investor funds at risk in smart Ponzi schemes reached nearly $3 billion in 2022. Such significant financial losses to investors seriously affect the development of the blockchain ecosystem [9], [10].

Although smart contract codes are publicly available on the blockchain, investors often struggle to verify them and end up trusting them blindly due to their lack of ability to thoroughly inspect the code [2]. To protect investors, various methods have been proposed to automatically detect smart Ponzi schemes [4], [6], [9], [11]–[16]. Early research focused on analyzing the transaction records on the blockchain and extracting features related to Ponzi schemes, such as transaction frequency, amount, and networks [11]–[14]. However, these approaches intrinsically depend on the transaction data, meaning that some investors have already finished the immutable transactions and suffered from financial losses. To achieve early detection before transactions occur, some approaches have focused on detecting Ponzi schemes using the smart contract code (Fig. 1A-C), including the source code, opcode, and bytecode [2], [4], [6], [9], [15], [17].

The above code-based approaches for smart Ponzi scheme detection still suffer from two major challenges: **(C1) limited adaptability:** These approaches inherently rely on fixed rules or models trained on existing labeled smart contracts. However, scammers continuously create new Ponzi scheme variants that evade these rules or criteria [10], making these ad-hoc approaches ineffective against new smart Ponzi schemes [15]. **(C2) lack of transparency:** These approaches leverage machine learning models, combined with abstract features of transaction records or codes, to classify smart contracts as smart Ponzi schemes without providing understandable and reliable evidence [7]. This makes it difficult for common investors to make confident investment decisions. Our preliminary work, *PonziLens* [17], attempted to visually explain Ponzi features at the opcode execution level, but it is still difficult for common investors to understand the complex opcode of smart contracts and a more intuitive approach is still missing [18].

To address these challenges, we propose ***PonziLens+***, a novel visual analytic approach to help smart contract investors and auditors identify smart Ponzi schemes intuitively. *PonziLens+* can reveal the semantic meaning of the smart contract execution process and allow users to conveniently validate various smart contracts across three levels: contract, execution path group, and execution path. Specifically, we first conduct a preliminary study with four domain experts to derive design requirements and propose a framework for semantic action extraction. This framework translates complex smart contract bytecodes into semantic action sequences, i.e.,

X. Wen and Y. Wang are with Nanyang Technological University. Email: {xiaolin.wen, yong-wang}@ntu.edu.sg. Part of this work was done when they were affiliated with Singapore Management University.

T. Nguyen, S. Ruan, J. Sun, and F. Zhu are with Singapore Management University. Email: {dtnguyen.2019, junsun, fdzhu}@smu.edu.sg, slruan.2021@phdcs.smu.edu.sg.

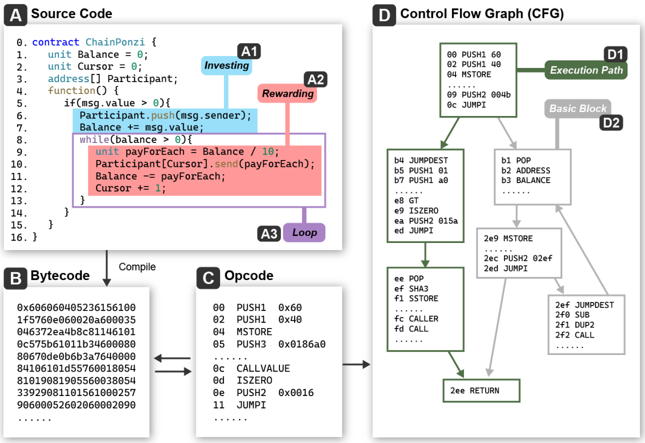
Q. Shen is with Southern University of Science and Technology. Email: shenqm@sustech.edu.cn.

Y. Wang and Q. Shen are the corresponding authors.

[1] https://ethereum.org/

a series of actions with meaningful semantics, to indicate the possible behaviors of a smart contract. These action sequences allow users to identify fraudulent behaviors and further verify whether a smart contract is a Ponzi scheme, instead of fully relying on fixed rules (**C1**). Furthermore, we propose a novel hierarchical visualization design including three modules to show the action sequences of a smart contract in a top-down manner, intuitively revealing the smart contract behaviors as well as the Ponzi scheme-related features (**C2**). With the help of *PonziLens+*, users can gain deep insights into the detailed behaviors of any smart contracts and identify smart Ponzi schemes intuitively and reliably. To evaluate the effectiveness and usability of *PonziLens+*, we conducted two case studies and in-depth interviews with domain experts and common investors. The results show that *PonziLens+* is effective for intuitively identifying smart Ponzi schemes. Additionally, *PonziLens+* also has potential applications in source code understanding [19] and software testing [20] beyond Ponzi schemes. To the best of our knowledge, we are the first to visualize smart contract codes for identifying smart Ponzi schemes. Our major contributions are listed as follows:

- We formulate the design requirements through collaboration with domain experts and propose a framework to extract semantic-meaningful actions to delineate the action sequences during the execution of smart contracts.
- We design *PonziLens+*, a novel visual analytic approach with three visualization modules to facilitate a comprehensive top-down analysis of Ponzi-scheme-related features in smart contracts, enabling adaptive and reliable identification of smart Ponzi schemes.
- We present two case studies and conduct in-depth user interviews with both domain experts and common investors to illustrate the effectiveness and usability of *PonziLens+*.



```
contract ChainPonzi {
   unit Balance = 0;
   unit Cursor = 0;
   address[] Participant;
   function() {
      if(msg.value > 0){
         Participant.push(msg.sender);
         Balance += msg.value;
         while(balance > 0){
            unit payForEach = Balance / 10;
            Participant[Cursor].send(payForEach);
            Balance -= payForEach;
            Cursor += 1;
         }
      }
   }
}
```

```
0x6060604052361561001
1f5760e060020a600035
046372ea4b8c81146101
0c575b61011b34600080
80670de0b6b3a7640000
84106101d55760018054
81019081905560038054
339290811015610002 57
9060005260206000 2090
......
```

```
00  PUSH1  0x60
02  PUSH1  0x40
04  MSTORE
05  PUSH3  0x0186a0
......
0c  CALLVALUE
0d  ISZERO
0e  PUSH2  0x0016
11  JUMPI
......
```

```
00 PUSH1 60
02 PUSH1 40
04 MSTORE
......
09 PUSH2 004b
0c JUMPI
```

```
b4 JUMPDEST
b5 PUSH1 01
b7 PUSH1 a0
......
e8 GT
e9 ISZERO
ea PUSH2 015a
ed JUMPI
```

```
b1 POP
b2 ADDRESS
b3 BALANCE
......
```

```
2e9 MSTORE
......
2ec PUSH2 02ef
2ed JUMPI
```

```
ee POP
ef SHA3
f1 SSTORE
......
fc CALLER
fd CALL
......
```

```
2ef JUMPDEST
2f0 SUB
2f1 DUP2
2f2 CALL
......
```

```
2ee RETURN
```

Fig. 1. The showcase of critical concepts in this study: (A) shows an example source code of a chain-type smart Ponzi scheme with investing behavior (A1) and rewarding (A2) behavior in a loop (A3). (B) and (C) show the bytecode and opcode of a smart contract. (D) demonstrates the control flow graph with execution paths (D1) and basic blocks (D2).

II. Related Work

This work is related to prior research on *smart Ponzi scheme detection*, *visual identification of blockchain frauds*, and *software visualization*.

A. Smart Ponzi Scheme Detection

Early studies on smart Ponzi scheme detection relied on manual checks of smart contracts' source codes [2]. Recently, automatic smart Ponzi scheme detection algorithms were proposed, which can mainly be divided into *transaction-based* and *code-based*, according to their inputs [15].

Transaction-based methods detect Ponzi schemes by extracting information from the existing transaction records of smart contracts. For instance, prior studies [11], [12], [14] have defined different features of transactions (e.g., payment frequency and balance) and further trained customized models with these features to automatically detect Ponzi schemes from the given smart contracts. Besides, Liang et al. [13] extracted the structure features of transaction networks and incorporated them in their Ponzi scheme detection model. These approaches intrinsically rely on transaction records, making them unable to achieve the early detection of Ponzi schemes before investors are really trapped in smart Ponzi schemes.

Code-based methods aim to detect smart Ponzi schemes before executing transactions by analyzing the code information (e.g., source code, opcode, and bytecode) available once the smart contract is deployed. Initially, these methods focused on relatively simple features like opcode frequency or source code text [2], [6], [21]–[23]. However, reducing intricate codes to simple features results in significant information loss, making it difficult to fully capture runtime information and rendering these methods unreliable for Ponzi scheme detection. Accordingly, recent studies have utilized *Control Flow Graphs (CFG)*, which graphically represent a smart contract's execution paths to capture its execution logic [9], [15], [24]. However, these approaches still lack an understandable explanation for why a smart contract is classified as a Ponzi scheme. Also, these methods are based on fixed patterns, rules, or models trained on existing labeled smart contracts, making it hard to detect new variants of Ponzi schemes. *PonziLens+* can visualize the bytecode execution process of a smart contract as semantically meaningful action sequences with highlighted Ponzi-scheme-related features. This allows users to deeply understand why a smart contract is a Ponzi scheme or not and identify new variants by checking the detailed behaviors.

B. Visual Identification of Blockchain Frauds

Due to the lack of effective regulation and complex data in blockchain application scenarios, numerous visualizations have been developed to help people understand blockchain data and detect fraud visually [25].

Most studies focus on fraud within blockchain transactions, identifying issues like coin mixing [26], stealing [27], and money laundering [28] by visualizing value flow and tracking cryptocurrency movement across entities over time. Other frauds can be recognized by visualizing features in the transaction networks, such as high-frequency transactions [29] and wash trading [30]. Also, visualizing the networks between various entities (e.g., addresses [31], clusters [32], and exchanges [33]) in the blockchain can help identify some abnormal structures and entities to avoid potential fraud.

In addition to frauds occurring in transaction records, there are also frauds deployed on the blockchain through smart contracts. These frauds can be detected before transactions occur by analyzing the code of smart contracts, including Ponzi schemes (the focus of this study). Before our study, there were no visualization tools designed for detecting fraud within smart contract codes. While there are existing visualizations aimed at aiding in understanding control flow graphs [34]–[36], none of them focus on fraud detection. Our preliminary study [17] aimed to visualize the execution paths in the original control flow graph and revealed the investment and rewarding flows by recognizing critical opcodes. Nevertheless, it is still difficult for common investors to understand opcodes and conduct an in-depth analysis of smart contract codes. Therefore, in this paper, we move one step further by extracting semantically-meaningful actions from the opcodes of smart contracts, which allows common investors to gain deep insights into smart contracts and easily identify smart Ponzi schemes.

C. Software Visualization

Smart contracts are software on the blockchain [37], which makes this work also relevant to software visualization. According to the visualized features of software systems, previous software visualizations fall into three categories [38], [39]: structure, evolution, and behavior. Structure visualizations aim to analyze information like source code [40], package dependencies [41], and control flow graphs [35], [36], to aid in understanding software structure. Evolution visualizations analyze the code change history to show the evolution of a software system [42], [43]. Behavior visualizations display data collected from program execution like function calls [44] to facilitate performance optimization [45] and anomaly detection [46]. Existing behavior visualizations typically display time series [47] or domain-specific event sequences [45], [48]. Our work belongs to the behavior visualization of software, as smart contract behaviors during execution are visualized. Unlike existing software behavior visualization methods, our approach introduces a novel hierarchical visual design that represents potential smart contract behaviors across three levels of granularity. This design emphasizes Ponzi scheme features at different levels of details, enabling users to efficiently audit a smart contract without navigating complex source code.

III. Background

This section introduces the background, including smart contracts, control flow graphs, and smart Ponzi schemes.

A. Smart Contracts

Smart contracts are self-executing programs deployed on a blockchain platform, which will be automatically executed when specific predefined trigger events are met. Smart contracts have been widely utilized across various domains, including automated payments and asset exchanges on decentralized finance (DeFi) platforms [49]. *Ethereum* [50] is one of the most well-known blockchain platforms with smart contracts incorporated. While other blockchains also support smart contracts, we focus on Ethereum in this study.

Smart contracts can be written in various programming languages like Solidity, Viper, and Serpent [51]. The code written in these languages is called the ***source code*** of smart contracts, and Fig. 1A shows an example of the Solidity source code of a smart contract. When deploying these smart contracts on the blockchain, they are compiled into the hexadecimal ***bytecode*** (Fig. 1B) and sent to the blockchain. Bytecode can be converted into a sequence of instructions written in ***opcode*** [52] (i.e., "*operation code*") and operands, as shown in Fig. 1C. For instance, the bytecode "*Ox6000*" is equal to an instruction with opcode "*PUSH1*" and operand "*0x00*". The bytecodes of all smart contracts are accessible on the blockchain, but not all source codes are publicly released. This is why we chose bytecodes as the input for our study.

B. Control Flow Graphs (CFG)

In Ethereum, smart contracts execute within the ***Ethereum Virtual Machine (EVM)***, a stack-based run-time environment. Similar to traditional programs, smart contracts use the ***stack*** for temporary data (e.g., holding variables during function calls), employ ***memory*** for short-term data storage within a transaction (similar to RAM, Random-Access Memory), and utilize ***storage*** for persistent data that remains accessible across transactions (like global variables) [50]. Most opcodes take the values at the stack top as operands and place the results back onto the stack. Among them, several opcodes can modify the memory (e.g., *MSTORE*) and storage (e.g., *SLOAD* and *SSTORE*) [52].

During actual execution in EVM, smart contracts execute different code paths based on conditional statements (e.g., *if* and *else*), loops, and function calls. ***CFG*** (Fig. 1D) is a static graphical representation of all potential execution paths (control flows) of a smart contract [53]. The smallest execution unit in CFG is the ***basic block*** (Fig. 1D2), which consists of a sequence of instructions. Each basic block has no loops, and the directed links between basic blocks indicate possible execution orders. When invoking a transaction, the contract executes along a sequence of basic blocks in the CFG, which is determined by the conditions. This sequence is called the ***Execution Path*** (Fig. 1D1).

C. Smart Ponzi Schemes

A *Ponzi scheme* is a classic financial scam in which investors are lured with promises of exceptionally high returns on their investments [1]. The scheme operates by using the investments from new investors to pay returns to earlier investors. Such a game continues until there are insufficient funds left, ultimately leading to the collapse [54]. A *smart Ponzi scheme* refers to a Ponzi scheme deployed on a smart contract [2], [4]. According to existing studies [2], [9], [15], the **features** of smart Ponzi schemes are as follows:

- **F1. Include *investing* and *rewarding* behaviors.** Investing indicates that after receiving an investment the contract records investment information like investor's ad-

dresses and investing amount. Rewarding means that the contract redistributes money among previous investors.
- **F2. Receive money only from investors.** This feature can rule out smart contracts with external asset sources, like a bank that pays the interest of a "smart" bond or enterprises that distribute incentives to employees.
- **F3. Guarantee higher profits for all participants.** Each investor can make a profit if there is enough money in the contract afterward. This rules out contracts like gambling or games, where not all investors can get high rewards.

Fig. 1A shows a smart Ponzi scheme written in Solidity. For F1, Lines 6-7 (Fig. 1A1) indicate that the contract stores the investor's address (*msg.sender*) into an array called *Participant* and updates the *Balance* with the investment amount (*msg.value*) under the condition that the investment amount is greater than zero (Line 5), which is an investing behavior. In Fig. 1A2, the contract uses a loop (A2) to pay 10% of the *Balance* to each previous investor, which is a rewarding behavior. This contract aligns with F2 as it contains only one function for receiving investments and distributing rewards to previous investors. If there are enough new investments, all the previous investors can get high rewards, satisfying F3. Therefore, this contract is indeed a smart Ponzi scheme.

IV. Informing the Design

To achieve a rational design, we conducted a preliminary study to gather the requirements for guidance. This section reports the details and feedback of our preliminary study.

A. Preliminary Study

During the preliminary study, we interviewed four domain experts from both academia and industry, with rich experience in smart Ponzi scheme detection. E1 and E2 are university professors whose research areas encompass web3 fraud detection and smart contract analysis. E3 works as a smart contract auditor at a web3 security company, while E4 serves as a business analyst in another web3 company with services in smart contract auditing. All four experts have strong backgrounds in smart contracts and diverse insights into smart Ponzi schemes. We conducted open-ended interviews with each expert individually, with each session lasting about one hour. The interview was constructed into three phases, each focusing on a specific aspect: *Necessity of Our Study*, *Ponzi Features at Bytecode Level*, and *Design Requirements*. The feedback about the necessity is introduced in the introduction and related work sections, while the Ponzi features and design requirements are described in the subsequent sections.

B. Ponzi Features at Bytecode Level

According to the previous studies [2], [15], smart Ponzi schemes can be roughly divided into four types:

- ***Handover-scheme***: Upon receiving investments from a new investor, the contract will proceed to send profits to the previous investor (the last one who has invested) and subsequently update the address of the new investor as the new "last investor" in the scheme.
- ***Chain-scheme***: Upon receiving investments from a new investor, the contract will distribute profits to all the previous investors stored in a chain-like structure. Then, the contract will add the new investor to this chain.
- ***Tree-scheme***: In a tree-scheme, investors are organized in a tree-like structure based on the invitation relationships. Upon receiving an investment, the contract will sequentially distribute profits to the previous inviters of each node in the tree.
- ***Withdraw-scheme***: In a withdraw-scheme, the amount of each investor's balance is stored in the blockchain. When investments are received, it will increase the balance of each investor, instead of the direct redistribution. Investors have to invoke another execution to withdraw their money according to their balance.

During interviews, domain experts pointed out that F1 plays a pivotal role in identifying Ponzi schemes in terms of contract bytecodes, i.e., the identification of investing and rewarding patterns, which helps eliminate non-Ponzi (not a Ponzi scheme) contracts quickly and distinguish the types of Ponzi schemes. Since F2 requires analyzing investing behaviors and F3 involves examining the amount of investing and rewarding, identifying these behaviors (F1) is fundamental to both F2 and F3. To facilitate the identification of F1, we define four critical features at the bytecode level based on the investing and rewarding behaviors of four typical smart Ponzi schemes, which are referred to as ***Ponzi Features (PF)*** in this paper.

- ***(PF1) Investing***: The execution path stores the investor's address in the EVM's storage for future reward payments.
- ***(PF2) Payment***: The execution path distributes funds to an entity. This Ponzi feature is helpful for the verification of fund destinations in smart contracts.
- ***(PF3) Loop***: A loop exists in the execution path. In chain-schemes and tree-schemes, loops iteratively process payments to previous investors, while in withdraw-schemes, loops update the balances of previous investors.
- ***(PF4) Rewarding***: The execution path directly redistributes investment to previous investors, demonstrated through payments, with the receiver retrieved from the same slots storing the investor addresses.

C. Design Requirements

During the preliminary study, all the experts (E1-E4) agreed that the visual identification of Ponzi schemes should follow a top-down workflow, beginning with a comprehensive understanding of the smart contract's functionality and eventually finding out the specific execution paths for conducting Ponzi schemes. Distilling experts' feedback, we organize the design requirements into three levels: **Contract**, **Group**, and **Path**. The contract level provides an overview of Ponzi features across all potential paths in a smart contract. The group level emphasizes action patterns within groups of execution paths of similar Ponzi features, while the path level concentrates on detailed execution information for each execution path.

- **R1** (Contract) **Overview the Ponzi feature distribution.** Experts (E1-E4) have noted that initiating the analysis by gaining an overview of the Ponzi feature distribution can

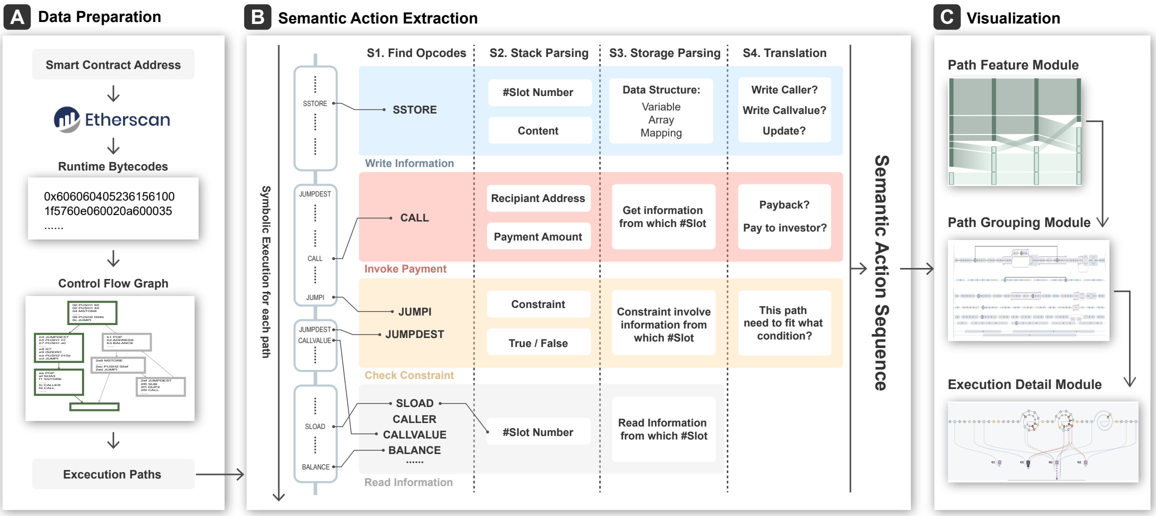


Fig. 2. The framework of *PonziLens+*. (A) shows the data preparation for the collection of potential execution paths. (B) shows the semantic action extraction that generates semantic action sequences from each execution path. (C) demonstrates three visualization modules in *PonziLens+*.

aid in rapidly assessing the possibility of a Ponzi scheme. Grouping paths with similar Ponzi features helps locate the paths responsible for conducting the Ponzi scheme.

- **R2** (Group) **Summarize action patterns within each execution path group.** All experts (E1-E4) mentioned that checking the action patterns of each path group can help quickly identify the functionality of the smart contract. E3 emphasized summarizing both unique and common actions among multiple execution paths can help understand the overall action patterns of each group more effectively than presenting each path individually.
- **R3** (Group) **Highlight the Ponzi features of each path group.** Given R2, two experts (E2, E4) commented that the visual summary should highlight Ponzi features upon the concrete actions, which can provide insight into the evidence of Ponzi schemes and help filter out critical execution paths for further analysis.
- **R4** (Path) **Show the detailed action sequences of individual paths during execution.** All experts (E1-E4) agreed that identifying a smart Ponzi scheme usually depends on the features in several paths, and it is necessary to analyze the actions of these paths during execution in detail. E3 pointed out that the time and conditions under which a path invokes a payment are crucial evidence for Ponzi scheme detection. Therefore, our method should visualize the whole action sequences of an execution path.
- **R5** (Path) **Support the deep analysis of action patterns of the loop.** All experts (E1-E4) emphasized the loop is a crucial Ponzi feature appearing in three out of the four Ponzi scheme types. Our method should facilitate the comprehension of behavior within each round of loops, including whether each round within the loop performs the same or different actions. Additionally, E2 added that actions within the loop can also help identify Ponzi scheme types and understand the money flow.
- **R6** (Path) **Display the storage interactions of individual paths during execution.** The storage stores permanent data of smart contracts, including investment information (i.e., previous investors' addresses and investment amounts) and rewarding information (i.e., recipient addresses and profit amounts). All experts (E1-E4) said that the storage interactions during execution reveal vital information for Ponzi scheme detection, such as which slots offer recipient addresses during payments. E4 added that the storage structure of investors' addresses indicates the Ponzi scheme types. For instance, a tree-like structure often indicates a tree-scheme, while the handover-scheme typically stores investors' addresses in variables [15].

V. Semantic Action Extraction

This section introduces the semantic action extraction component of our approach (Fig. 2B).

A. Action Definitions

As shown in Fig. 1, the bytecodes of smart contracts are abstract and complex for common investors to understand. Therefore, we define a set of critical and semantic-meaningful actions related to Ponzi schemes and transform the bytecodes of smart contracts into meaningful action sequences. The definitions of actions ensure that the aforementioned Ponzi features can be reflected and the design requirements can be supported by these actions. Specifically, we define four critical actions related to smart Ponzi schemes. The actions and corresponding definitions are as follows:

Write Information captures the action of storing information in EVM storage slots. This action can help capture PF1, i.e., storing investment information, as well as inferring the contents and structure of storage slots (R6).

Invoke Payment signifies that a smart contract issues a specific payment to a recipient, a common feature (PF2) in all the Ponzi schemes for distributing profits to investors. Tracking money flow is crucial for identifying such schemes.

Check Constraint represents the specific conditions that the execution path should meet, particularly at decision points in CFGs. For example, the path involving the investing function requires that the amount invested must exceed a specific value.

Read Information involves loading values into EVM stacks for processing, including data from storage slots or EVM context (like contract balances or current timestamps). While this action is not tied to a specific Ponzi scheme, it aids in deducing a complete action sequence, particularly when other actions are too complex to understand.

B. Data Preparation

To identify and generate the above four actions, we first collect all feasible execution paths of a smart contract as depicted in Fig. 2A. Given a smart contract address that users want to analyze, we start by retrieving the bytecode using EtherScan[2], a well-known blockchain explorer. Subsequently, we build a CFG from these bytecodes with the aid of Teether, a smart contract testing tool [55], and traverse all paths in the graph. According to R5, when a loop is present in a path, we collect only the paths that encompass two rounds of the loop, which reduces the repeated paths and ensures sufficient information is gathered to deduce the loop's functionality. Finally, we use Teether [55] to run symbolic execution and test each path, collecting all potential execution paths that can successfully run on the real blockchain.

Symbolic execution is a software testing technique that replaces normal inputs (e.g., numbers) with symbolic values (e.g., formulae) during execution [56]. For smart contracts, actual values are replaced by symbolic ones, such as *CALLER* for the investor's address and *CALLVALUE* for the investment amount. During the symbolic execution, the contents in the stacks are represented as Z3 constraints, i.e., formulas with semantic meanings that can be parsed by Z3 [57], a constraint solver. For example, if the top two stack contents are *CALLVALUE* and *1*, executing the *ADD* opcode will result in a Z3 constraint *CALLVALUE+1*. By analyzing these Z3 constraints in stacks when specific opcodes are encountered, we can derive the semantic meaning of the opcode operations and construct the corresponding semantic actions.

C. Action Sequence Generation

We select a subset of opcodes related to the four actions and construct a semantic action sequence for each collected path. Specifically, we extract actions by analyzing the content in the EVM when encountering specific opcodes during the above symbolic execution, following four steps (S1-S4) shown in Fig. 2B. Steps S1-S3 follow the bytecode execution process on the EVM, where each opcode first manipulates the stack and then interacts with storage, while S4 aims to make the generated actions understandable for users.

S1. Find Specific Opcodes: Each action is associated with specific opcodes. For example, *SSTORE* is used for Ⓦ*Writing Information*, while *CALL* is used to Ⓟ*Invoke Payment*. Upon encountering any of these opcodes during symbolic execution, we can initially identify the four action types ⓌⓅⒸⓇ and then conduct S2-S4 to enrich its semantic information.

S2. Stack Parsing: In the EVM stack, we extract the current operation parameters for the above opcodes. For example, the two parameters of *SSTORE* indicate what content ● is stored in which storage slot for Ⓦ*Write Information*. With *CALL*, we identify the recipient's address and the payment amount for Ⓟ*Invoke Payment*. These parameters are represented as symbols in the format of Z3 constraints [57] during symbolic execution, instead of concrete numerical values, and are interactively shown in the tooltip in *PonziLens+*.

S3. Storage Parsing: By analyzing these Z3 constraints collected in S2, we can collect the relationships between storage slots and actions, i.e., where these values are collected from or related to which slots, and the slot structures. (The relationships between storage slots and actions are represented as the links between actions and storage slots in the Execution Detail Module). For instance, Ⓟ*Invoke Payment* retrieves the recipient's address from which slot or calculates the payment amount depending on values from which slot. For Ⓦ*Write Information*, we can recognize the data structures (e.g., variable ■ or array ▤) of storage slots by the different Z3 constraints for slot numbers.

S4. Translation: With S1-S3, we can depict actions from the perspective of program execution, such as writing specific content into a particular slot. S4 aims to translate the information collected from S1-S3, especially Z3 constraints, into understandable semantics, further aiding users in identifying Ponzi scheme features. For example, if the content written by *Write Information* is identified as the investor's address, we mark it as ◉*Investing (PF1)* and record the target slot storing investor addresses. Its slot structure can help identify the chain or tree types of Ponzi schemes. If an ◉*Invoke Payment* action collects the payee's address from this slot, it is marked as ◉◉*Rewarding (PF4)*. Additionally, if *Write Information* writes content that includes the previous content of the same slot, we mark it as "Ⓦ*Update Information*", commonly used in Ponzi schemes for updating investor counts or balance. If *Invoke Payment*'s payee is the current investor *CALLER* ⊕, we label it as "Ⓟ*Payback*", which often occurs when investors withdraw money, indicating a withdraw-scheme, or return their investment when certain conditions are not met.

Through these four steps, we collected action types (S1), operands (S2), storage slot interactions (S3), and relations with Ponzi features and additional semantics (S4), representing each execution path as a sequence of semantic actions.

VI. *PonziLens+*

Built upon the semantic action sequences, we propose *PonziLens+*, a visual analytic system to facilitate the early identification of smart Ponzi schemes. As shown in Fig. 2C, *PonziLens+* includes three visualization modules that enable users to diagnose a smart contract at three levels. *Path Feature Module* displays the distribution of Ponzi features across all the paths by grouping paths with similar features (R1). *Path Grouping Module* offers visual summaries for each path group (R2) and highlights Ponzi features within it (R3). *Execution Detail Module* presents detailed action sequences and storage interactions during individual executions (R4-R6). Fig. 3 shows the user interface of *PonziLens+*, where users initially see the Path Feature Module and Path Grouping Module and

[2] https://etherscan.io/

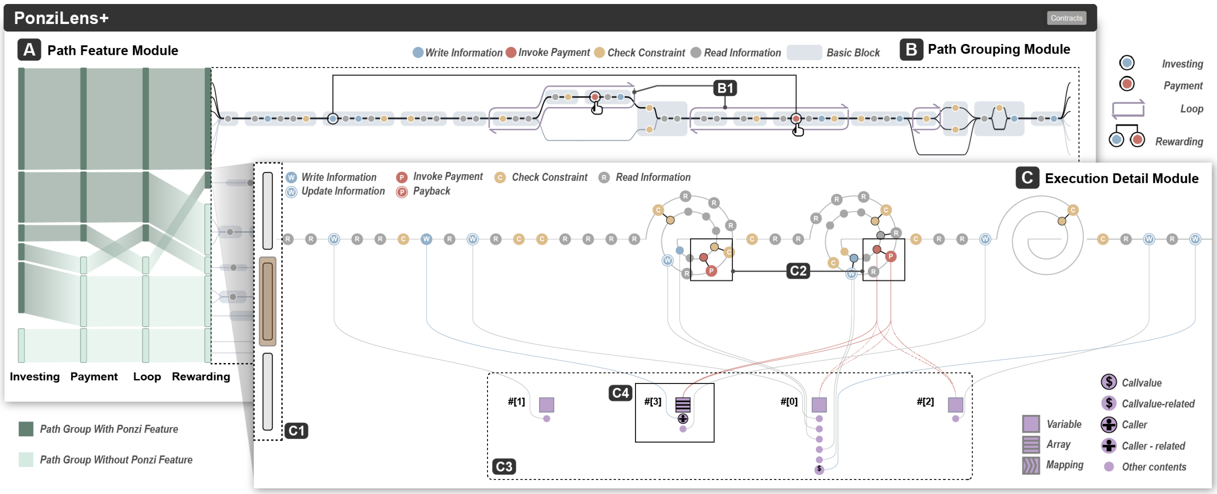


Fig. 3. The *PonziLens+* interface initially presents the Path Feature Module (A) and Path Grouping Module (B). Upon selecting specific actions of interest, users can access the Execution Detail Module (C), including a scroll bar (C1) to allow users to delve into one single path for more details.

can unfold the Execution Detail Module by selecting one or multiple paths from the Path Grouping Module.

A. Path Feature Module

The Path Feature Module (Fig. 4A) offers a quick overview of Ponzi feature distribution across all the execution paths, assisting users in identifying path groups that have a high risk of being a Ponzi scheme. Initially, the execution paths are grouped based on the four Ponzi features (**PF1-PF4**) mentioned in Section IV-B (Fig. 4A1), with paths sharing the same features grouped. PF1 and PF4 are marked in S4 of semantic action extraction while PF2 and PF3 are labeled based on the presence of *Invoke Payment* and loops in the action sequences. To visualize the distribution of multiple dimensions (PFs) across various path groups, parallel sets offer a natural visual design since it has been proven effective in visualizing categorical data distribution across multiple dimensions [58]. Parallel sets can also clearly demonstrate the number of paths in each group, which is crucial for users to estimate the proportion of paths with suspicious features. Moreover, encoding the number of paths in each group through band width helps link to the Path Grouping Module, allowing users to smoothly delve into path behaviors. Therefore, we adopt parallel sets as the layout for the Path Feature Module. Specifically, each column corresponds to a Ponzi feature. Path groups are represented by bands (Fig. 4A3), with their widths (Fig. 4A2) reflecting the number of paths in each group. Path groups with a certain feature are indicated in dark green ■, while those without are shown in light green □. These bands traverse four columns and are colored based on whether the groups possess these features. The total height of the dark (Fig. 4A4) or light green (Fig. 4A5) bars represent the count of paths with or without certain Ponzi features, respectively.

The arrangement of both Ponzi features and path groups is important in the Path Feature Module since a bad arrangement will make the analysis counter-intuitive or result in band crossings [58]. We ordered the Ponzi features according to the typical flows of manual analysis, starting with "Investing" and followed by "Payment" to identify direct redistribution after receiving investments. "Loop" checking is used next to eliminate irrelevant paths. "Rewarding", indicative of paying previous investors, is listed as the final feature, as it needs the first two features and helps determine which group has a high risk for subsequent analysis. To reduce edge crossing, we first set an initial order by sorting path groups based on whether they contain the four features. Groups with a specific feature are placed above those without it. In practice, users can also interactively adjust the feature order according to their interests by dragging and dropping the Ponzi feature name.

B. Path Grouping Module

The Path Grouping Module (Fig. 3B) offers visual summaries for each group, where we use circles in four diverse colors to represent four semantic actions (i.e., blue for "● *Write Information*", red for "● *Invoke Payment*", yellow for "● *Check Constraint*", and grey for "● *Read Information*"), as shown in Fig. 4B. The action sequences are arranged linearly from left to right, encoding the execution order, and are divided according to their basic blocks, with grey blocks □ added behind them for clarity and distinction.

Since displaying all paths is overwhelming, we propose a two-step path-merging strategy (Appendix A) to maintain the essential backbone of paths within a single group. First, we group paths that share the same basic block sub-sequence without order conflicts and merge paths based on those basic blocks, as shown in Fig. 4B2. Second, we check each merged basic block to verify whether all paths containing this block have the same action sub-sequence, and further separate the paths with different actions within the merged blocks, as shown in Fig. 4B3. Note that actions of the same type but with different parameters (operands in the stacks during symbolic execution) are also separated. When drawing this visualization, the x-axis represents the basic block sequence of merged paths, covering all blocks traversed by paths in this group. The y-axis marks the start and end points of all paths in the group before merging. In the middle, the width of each line

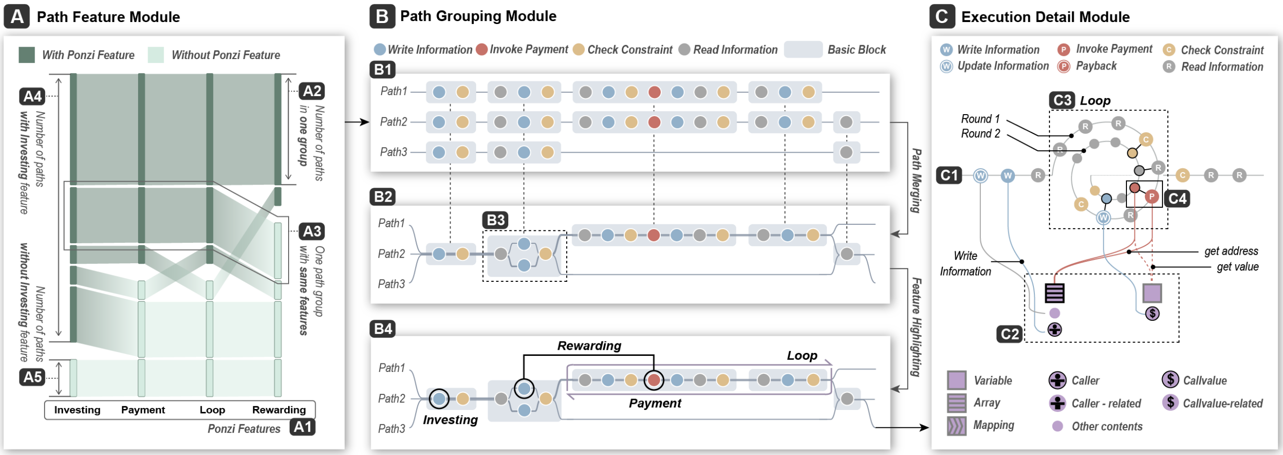


Fig. 4. The visual design of three visualization modules in *PonziLens+*. Path Feature Module (A) shows the Ponzi feature distribution across all execution paths. Path Grouping Module (B) provides a visual summary of action patterns in each path group, incorporating a path merging strategy (B1-B3) and highlighted Ponzi features (B4). Execution Detail Module (C) shows the detailed action sequences and storage interactions in each execution path.

reflects the number of merged paths. Therefore, this module can provide a comprehensive summary of the action patterns within each group, offering a clear and organized view of the involved execution paths (R2). Also, we highlight the Ponzi features (S4 in Section V-C) in the action summary (R3) to distinctly mark where these features occur during execution, as shown in Fig. 4B4. The “◉Investing” and “◉Payment” are highlighted by black circles surrounding blue and red circles, respectively. “Loop” is represented by a purple wrapper ⊏ that encases the basic blocks involved in the loop. To signify the “rewarding”, which means paying previous investors, we connect the “investing” and “payment” actions that operate on the same storage slot with a black line ◉◉.

C. Execution Detail Module

The Execution Detail Module (Fig. 4C) displays the detailed action patterns of individual paths (R4). Actions are placed from left to right, like those in the Path Grouping Module, as shown in Fig. 4C1. For clarity, it uses large icons with a letter inside to represent the four actions Ⓦ Ⓟ Ⓒ Ⓡ, which are encoded in the same color scheme as those in the Path Grouping Module. “Ⓦ*Update Information*” and “Ⓟ*Payback*” (S4 in Section V-C) are highlighted with a white border.

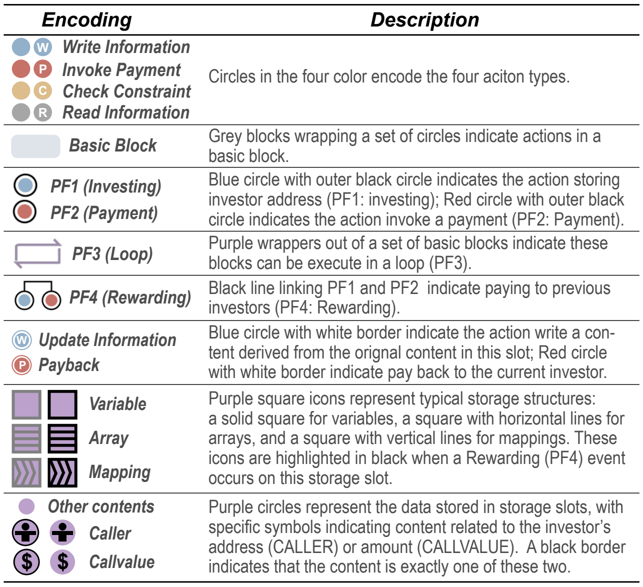
For an in-depth analysis of loops, we collect actions from two rounds of each loop for comparison (R5), since one round is not enough to confirm whether the loop repeats the same actions across different rounds. Two rounds are usually adequate for users to spot possible changes, making it unnecessary to include more rounds. Specifically, the actions within loops in the Execution Detail Module are arranged on a two-round Archimedean spiral (Fig. 4C3), with the two rounds placed on the outer and inner circles, respectively. The Archimedean spiral ensures uniform distance between the same actions of two rounds, facilitating an easy comparison. The spiral's design aligns with the user's intuitive understanding of loops. As loops typically traverse the same basic blocks, the two rounds often involve the same sequence of actions, though the same actions may have different parameters. The actions in the second round (inner circle) are simplified as smaller circles without letters to reduce repeated information. Those actions with different parameters in the second round are highlighted with black circles and middle lines, as illustrated in Fig. 4C4, which aids users in recognizing the differences quickly.

For visualizing storage interactions (S3 in Section V-C), the storage slots used by the smart contract are depicted beneath the action sequence (as shown in Fig. 3C3). Three distinct purple square icons represent the typical storage structures: a pure square for variables ■, a square with horizontal lines for arrays ▤, and a square with vertical lines for mappings ▥. Purple circles ● are listed under these square icons to symbolize the data stored in these slots, where we use some symbols (as shown at the bottom of Fig. 4C) to highlight the content involving the Ponzi-scheme-related information like the investor's address (*CALLER* ⊕) and amount (*CALLVALUE* $). The contents stored in these slots are connected to the corresponding actions in the action sequence that writes them. These connections are represented by lines, colored blue for writing Ponzi-scheme-related information and grey for writing unrelated information. In addition, red solid and dashed lines are used to indicate which storage slots the “Ⓟ*Invoke Payment*” retrieves its parameters from, for the recipient address and payment amount, respectively. When an action writes *CALLER* in a slot and a payment action retrieves the recipient's address from the same slot, this slot is highlighted in black to denote “Rewarding” previous investors directly since this slot is used for storing investors' addresses in this smart contract. In summary, the Execution Detail Module enables users to infer the structure and functionality of each storage slot and to understand the storage interactions of the “Ⓦ*Write Information*” and “Ⓟ*Invoke Payment*” actions.

D. Interactions

PonziLens+ enables rich interactions to allow users to verify whether a smart contract is a Ponzi scheme smoothly.

Hovering to show details. In *PonziLens+*, users can hover over an action to view a tooltip displaying details of this action collected from S1-S4 in Section V-C, such as target slot and content for *Write Information*, payee and value for



Encoding	Description
● Ⓦ Write Information ● Ⓟ Invoke Payment ● Ⓒ Check Constraint ● Ⓡ Read Information	Circles in the four color encode the four aciton types.
▭ Basic Block	Grey blocks wrapping a set of circles indicate actions in a basic block.
◉ PF1 (Investing) ◉ PF2 (Payment)	Blue circle with outer black circle indicates the action storing investor address (PF1: investing); Red circle with outer black circle indicates the action invoke a payment (PF2: Payment).
⟲ PF3 (Loop)	Purple wrappers out of a set of basic blocks indicate these blocks can be execute in a loop (PF3).
◉◉ PF4 (Rewarding)	Black line linking PF1 and PF2 indicate paying to previous investors (PF4: Rewarding).
Ⓦ Update Information Ⓟ Payback	Blue circle with white border indicate the action write a content derived from the orignal content in this slot; Red circle with white border indicate pay back to the current investor.
■ ■ Variable ☰ ☰ Array ▨ ▨ Mapping	Purple square icons represent typical storage structures: a solid square for variables, a square with horizontal lines for arrays, and a square with vertical lines for mappings. These icons are highlighted in black when a Rewarding (PF4) event occurs on this storage slot.
● Other contents ● ● Caller $ $ Callvalue	Purple circles represent the data stored in storage slots, with specific symbols indicating content related to the investor's address (CALLER) or amount (CALLVALUE). A black border indicates that the content is exactly one of these two.

Fig. 5. A summary of visual encoding used in *PonziLens+*.

Invoke Payment, and specific constraints for *Check Constraint*. For storage contents, hovering reveals the underlying Z3 constraints that indicate the content's meaning.

Navigating to paths of interest. *PonziLens+* enables users to easily locate suspicious paths among numerous paths by interacting with three modules. Users can first select a path group with specific Ponzi Features in the Path Feature Module, then click on actions of interest with specific patterns in the Path Grouping Module. These actions are highlighted with a hand icon, and all related paths are marked in black for easy tracking. Meanwhile, the Execution Detail Module appears below to display detailed execution information for the paths containing the selected actions, and users can switch between paths using a scroll bar (Fig.3C1).

Comparing two rounds in a loop. When users want to check the changes between two rounds of a loop, they are allowed to click the highlighted actions in the inner circle to invoke the tooltip to demonstrate the differences in the parameters between the two rounds, where the same parts are colored in green while the different parts are colored in red.

The case studies in Section VII will show how to use these interactions in *PonziLens+*.

VII. Case Study

This section presents two case studies to demonstrate the effectiveness of *PonziLens+* in identifying smart Ponzi schemes, with or without typical features. These case studies were conducted by two users (U4 and U7) of our user interviews, as will be introduced in Section VIII.

A. Case 1: Scrutinize a Typical Chain Scheme

U7 is the creator of a Web3 community and has four years of Web3 investment experience. He knows what smart Ponzi schemes are, but lacks experience in auditing the source code. In the Path Feature Module, U7 quickly identified a path group possessing all four Ponzi features (Fig. 3A), indicating that each path in this group involves investing, rewarding, and at least one loop. Next, he investigated the visual summary of this group (Fig. 3B) and noticed two purple wrappers with a red circle highlighted with a black circle◉, as depicted in Fig. 3B1. It showed that the path group contained two loop-involved payments. Further, a red circle in the loop connecting to a blue circle by a black line ◉◉suggests repeated payments to earlier investors, which is a clear sign of a Ponzi scheme. U7 wondered if the two loops conducted repeated actions, so he clicked to select two payment actions in the loops and unfold the Execution Detail Module. As shown in Fig. 3C2, he easily observed that payments in the inner circle were highlighted in both loops, signifying that this path invokes different payments in two rounds of loops. He also noted that the payments in the first loop had no connections to the storage, whereas those in the second loop retrieved the receiver's address and the payment amount from storage. From this, he speculated that the payments of the first loop probably were not to previous investors, while the payments in the second loop were mainly for distributing money to existing investors.

The chain-scheme is one of the four popular types of smart Ponzi schemes [15], utilizing a loop to redistribute new investments to each previous investor, whose addresses are stored in a chain-like structure (e.g., an array). Given that this smart contract involves payments to previous investors in a loop and the slots storing investors' addresses are exactly an array (Fig. 3C4), U7 expressed strong confidence in concluding that this contract is indeed a smart Ponzi Scheme, i.e., a chain-scheme. Such a judgment helped him with an easy decision-making of not investing in this smart contract.

B. Case 2: Identify a new variant of smart Ponzi Scheme

U4 is a smart contract auditor at a Web3 company and one major task of his daily job is to detect vulnerabilities in smart contracts through programs and manual inspection. He knows smart Ponzi schemes and the losses they cause to investors, but his work focuses on checking whether smart contracts can be executed successfully instead of frauds like Ponzi schemes.

Initially, U4 observed that there were no path groups with multiple Ponzi features and only one path group has the payment feature in the Path Feature Module (Fig. 6A). Thus, he initially speculated that this smart contract was probably not a smart Ponzi scheme. But to further validate his judgment, he chose to delve deeper and checked the Path Grouping Module (Fig. 6B). He noticed that the paths in the group with payments were divided into two subgroups by our path-merging strategy. By hovering over the two yellow circles (i.e., "●*Check Constraint*"), U4 noticed that both groups had the same constraint (*CALLVALUE>1000Wei*) but different conditions, i.e., *False* for the upper one (Fig. 6B1) and *True* for the lower one (Fig. 6B2). This indicated that if the investment amount was more than 1000 Wei (a measurement unit of the native Ethereum cryptocurrency), the contract would execute paths in the lower subgroup, and vice versa. After clicking on the payment action in Fig. 6B3 to view the execution details, U4 noticed a red circle enclosed by white Ⓟ and connected to Slot 2 via a dashed line. This indicated that the contract returned

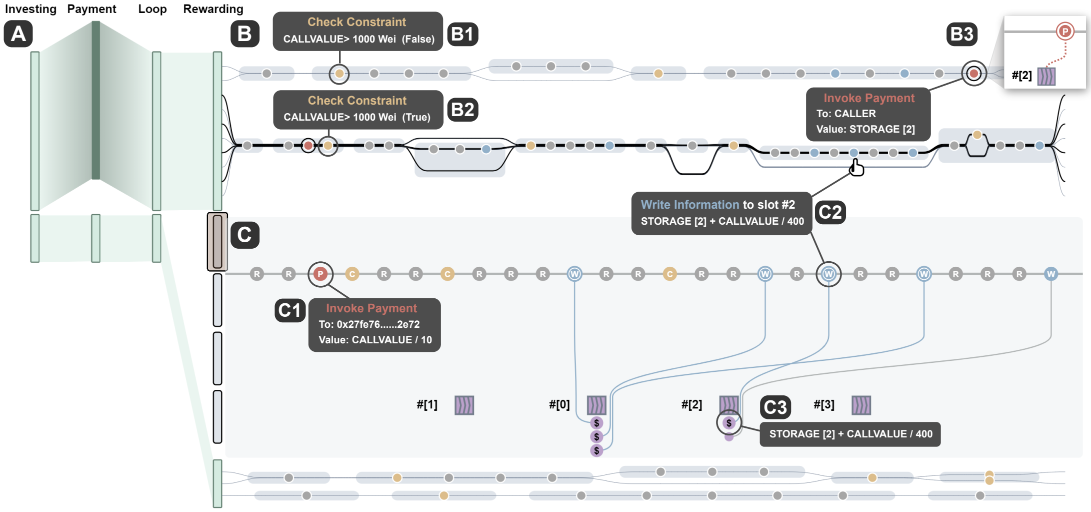


Fig. 6. A case for identifying a non-typical smart Ponzi scheme. The Path Feature Module (A) shows no path groups with multiple Ponzi features. However, the Path Grouping Module (B) shows subgroups of paths with different investing amounts (B1 and B2), and one subgroup pays back the money to the *CALLER* (B3). The Execution Detail Module (C) shows more execution details to help conclude this contract is a Ponzi scheme.

the money to the current *CALLER*, with the payment amount retrieved from Slot 2. U4 confirmed that the paths in this subgroup were probably used to invoke a withdrawal function, returning money to the current *CALLER*. Being curious about how the withdrawn amount was determined, he checked each blue circle to find the "●*Write Information*" that stored the amount in Slot 2 (Fig. 6C2) and clicked it to view the details in the Execution Detail Module (Fig. 6C). The content stored in Slot 2 (Fig. 6C3) showed that the amount that investors can withdraw was increased by *CALLVALUE/400*. Since these paths were executed only when the investment exceeded 1000 *Wei*, it indicated that the value would increase with each new investment. Additionally, U4 noticed a "P*Invoke Payment*" (Fig. 6C1) at the start of the path, where *CALLVALUE/10* was paid to a specific address. He concluded that the smart contract charged ten percent on each investment.

The above analysis overturned U4's initial hypothesis, leading him to conclude that the contract was indeed a smart Ponzi scheme, despite lacking typical features like loops or direct rewards. This contract still exhibited essential Ponzi scheme characteristics, such as maintaining a record of funds owed to prior investors, increasing profit amount based on a portion of new investments, and enabling investors to withdraw profits through additional invocations of the smart contract. U4 noted that when the contract balance is sufficient, all investors can receive profits, while the scheme would collapse once the balance becomes limited and unable to sustain payouts to prior investors. He also emphasized that this kind of contract can easily evade existing rule-based detection methods and that *PonziLens+* was indeed effective in identifying such a new variant of those typical smart Ponzi schemes.

VIII. User Interview

We conducted semi-structured user interviews with 12 target users to evaluate the effectiveness and usability of *PonziLens+*.

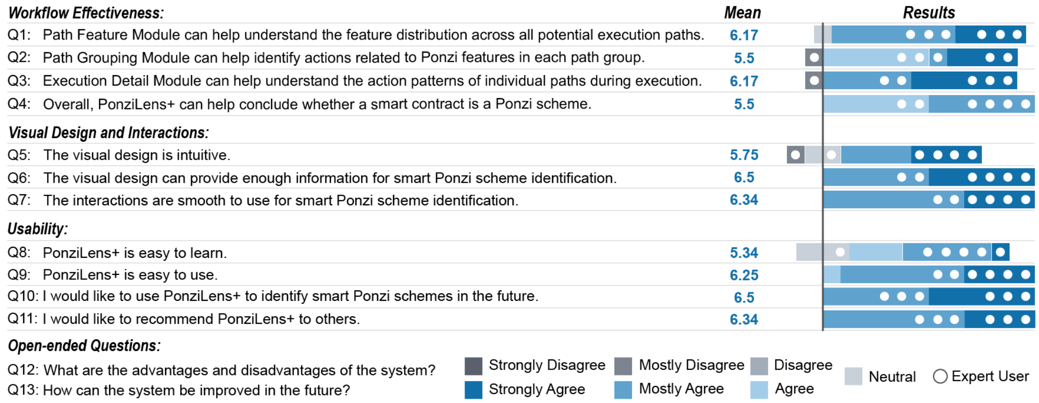
A. Participants and Apparatus

We recruited 12 participants (U1-U12) from Web3 communities and universities for our user interviews (2 females, 10 males, $age_{mean} = 28.33$, $age_{sd} = 4.21$, with normal vision and no color-blindness). All the participants have enough background in blockchain and smart contracts, and they have experience in investing in smart contracts. Our participants can be categorized as expert users and common users based on their smart contract auditing experience. U1-U6 are experts skilled in writing and auditing smart contract source code. U7-U12 are typical investors without such auditing experience, with only U7 being able to understand source code. Among them, five participants (U1-U3, U6, U8) are academic researchers specializing in Web3 security, with two (U2-U3) focusing on smart Ponzi scheme detection. U4 and U5 have experience in smart contract auditing within Web3 security companies. The other participants (U7, U9-U12) are typical investors with domain knowledge. Participants' profiles are shown in Table 1 of Appendix B.

Our interviews were online via Zoom. We launched the prototype system of *PonziLens+* on the server and allowed participants to assess it via their own laptops or desktops. Each interview lasted about one hour, and we paid a compensation of $15 to each participant for their time in our user interviews.

B. Procedure

The interview began with an explanation of the background, visual design, and workflow of *PonziLens+*. Following this, we presented a usage scenario to the participants, guiding them on how to utilize *PonziLens+* for verifying a smart contract. The tutorial above lasted about 15 minutes. During the interview, the only task was to check whether a smart contract was a Ponzi scheme. Hence, we asked participants to leverage *PonziLens+* to verify two smart contracts: one Ponzi contract and one non-Ponzi contract, both randomly selected from a published dataset [10]. This task phase had no hard time limit and lasted until participants reached a conclusion,



	Mean
Workflow Effectiveness:	
Q1: Path Feature Module can help understand the feature distribution across all potential execution paths.	6.17
Q2: Path Grouping Module can help identify actions related to Ponzi features in each path group.	5.5
Q3: Execution Detail Module can help understand the action patterns of individual paths during execution.	6.17
Q4: Overall, PonziLens+ can help conclude whether a smart contract is a Ponzi scheme.	5.5
Visual Design and Interactions:	
Q5: The visual design is intuitive.	5.75
Q6: The visual design can provide enough information for smart Ponzi scheme identification.	6.5
Q7: The interactions are smooth to use for smart Ponzi scheme identification.	6.34
Usability:	
Q8: PonziLens+ is easy to learn.	5.34
Q9: PonziLens+ is easy to use.	6.25
Q10: I would like to use PonziLens+ to identify smart Ponzi schemes in the future.	6.5
Q11: I would like to recommend PonziLens+ to others.	6.34

Open-ended Questions:

Q12: What are the advantages and disadvantages of the system?

Q13: How can the system be improved in the future?

Fig. 7. The user interview questionnaire results. Q1-Q11 are close-ended questions assessing *PonziLens+*'s workflow effectiveness (Q1-Q4), visual design and interactions (Q5-Q7), and usability (Q8-Q11), rated on a 7-point Likert scale. Q12-Q13 are open-ended, collecting participant feedback on pros and cons. Results are visualized in a horizontally stacked bar chart, with expert users marked by white circles.

which usually lasted about 30 minutes in practice. Finally, we asked participants to finish a post-study questionnaire with 13 questions (Q1-Q13), as shown in Fig.7. Q1-Q11 are close-ended questions that should be answered on a 7-point Likert scale and are designed to evaluate *PonziLens+*'s workflow effectiveness (Q1-Q4), visual design and interactions (Q5-Q7), and usability (Q8-Q11). Q12-Q13 are open-ended questions to collect participants' feedback on the advantages/disadvantages and possible improvements of *PonziLens+*. Overall, each user interview session took about 60 minutes. All the data collected from the participants were recorded with their permission.

C. Results

Fig. 7 presents the participant responses from our post-study questionnaire. Overall, *PonziLens+* received high ratings for the close-ended questions. Participants unanimously confirmed that there were no existing tools like *PonziLens+* that could offer convincing rationales for identifying smart Ponzi schemes. Notably, expert users tended to give higher scores, appreciating *PonziLens+*'s effectiveness in benefiting their contract auditing, as marked in Fig. 7. However, common investors desired more comprehensive tutorials, as they often have a limited background in scrutinizing smart contract execution. The main feedback can be summarized as follows:

Workflow Effectiveness. Most ratings for Q1-Q4 are positive, which demonstrates that *PonziLens+* has an effective workflow for the visual identification of smart Ponzi schemes. Notably, U5 scored "2" for Q2 and Q3. He faced challenges in correlating the actions with the smart contract's source code, as his work only involved source code analysis. U5 acknowledged that symbolically executing bytecodes accurately reflects the actual execution process and can cover all smart contracts on the blockchain. However, he pointed out that investors typically lack trust in smart contracts without available source codes, so the source codes can also become suitable inputs to offer extra information for verification. Among the three visualization modules (Q1-Q3), the Path Grouping Module received a relatively low mean score (i.e., $5.5/7$). U8, U9, and U11 found it easy to identify actions marked as Ponzi features, like those in loops or black outer circles, but struggled to assess the usefulness of unhighlighted actions, such as *Check Constraint*, in identifying Ponzi schemes. Therefore, they requested specific examples of unhighlighted actions in tutorials for better guidance.

Visual Design and Interactions. All participants agreed that our visual designs are intuitive and provide enough information for identifying Ponzi schemes. Also, they found the interactions user-friendly. They appreciated the Path Feature Module and highlighted features in the Path Grouping Module, which can help quickly understand potential suspicious behaviors. However, some found *PonziLens+* hard to grasp at the very beginning, particularly in establishing the initial correlation between execution processes and visual encoding. Once they grasped our semantic action scheme, they acknowledged the design's clarity in showing execution processes.

Usability. Overall, participants believed that *PonziLens+* was easy to learn and use (Q8-Q9). After familiarizing themselves with *PonziLens+*, they all expressed willingness to use it in the future and recommend it to others (Q10-Q11). Both expert users and common investors noted that it took them some time to learn the typical features and action patterns of various Ponzi scheme types due to the lack of knowledge about them. They suggested that more detailed tutorial documents and examples could be included in *PonziLens+*.

Open-ended Questions. In response to Q12, all the participants praised *PonziLens+* for offering clear evidence of Ponzi scheme detection, which boosted their analysis confidence. Additionally, visualizing all potential behaviors greatly aids in understanding the contract's function. However, they expressed concerns about the learning curve; while investors can easily grasp actions like payments, they may struggle with concepts like storage interaction and stack manipulation. Despite not fully understanding the underlying execution of smart contracts, participants indicated that *PonziLens+* helps them correlate actions with Ponzi features. For future improvements (Q13), three participants (U2, U6, and U11) proposed adding an initial suspicious score for smart contracts to guide subsequent analysis in *PonziLens+*. U3 pointed out that existing Ponzi features and types might not encompass all new Ponzi schemes. He suggested that *PonziLens+* could be enhanced to allow users to define new features and incorporate them into the Path Feature Module. These suggestions are promising and we plan to further improve *PonziLens+* according to them.

IX. Discussion

In this section, we report the lessons we learned during the development of *PonziLens+*. Also, we discuss the generalizability and limitations of *PonziLens+*.

Algorithms vs. Visualization. Before our work, smart Ponzi scheme detection mainly relied on manual inspection and automatic algorithms [10]. The advantage of algorithms is their ability to quickly scan numerous contracts without human effort. However, they depend on manual labels or predefined rules, which may not cover all Ponzi scheme types and can also result in false alarms. During this study, *PonziLens+* has been utilized to check some smart contracts labeled by automatic algorithms in prior research [15] and have found some wrong labels provided by the automatic algorithms. One example has been reported in Section VII-B). Also, we have identified some smart Ponzi schemes that lack typical features or can not be classified into any known Ponzi scheme types. Further, investors prioritize precise, understandable results for specific smart contracts of interest, rather than checking a large amount of smart contracts at once. In this context, visualization can play a critical role in revealing the evidence of Ponzi schemes intuitively and additionally aids in both manual inspections and expanding labeled datasets rapidly.

Experts vs. Investors. *PonziLens+* is designed for both smart contract auditors and common investors hoping to make informed investment decisions. Our user interviews revealed different analytical ways between these two types of users. Common investors particularly appreciate the Path Feature Module and Path Grouping Module, which enable them to observe the Ponzi feature distribution and check how these features manifest in action sequences by the highlighted Ponzi features. They usually do not delve into the Execution Detail Module to check the storage interactions, except for checking the loop details. They also recommend adding an initial suspicious score and more examples of action patterns in various Ponzi schemes. However, expert users prefer to scrutinize detailed action patterns in both the Path Grouping Module and Execution Detail Module to gather more convincing evidence during execution because they find the patterns in the Path Feature Module a bit general to reach a solid conclusion about whether it is a smart Ponzi scheme. They also desire to incorporate more information about the source code for mapping action patterns to the actual contract codes. Overall, *PonziLens+* meets the needs of experts and investors, but it can be improved to further satisfy their customized requirements.

Generalizability. Although *PonziLens+* focuses on smart Ponzi schemes, it can be generalized to other applications. First, our workflow and visual designs can be easily extended to identify other vulnerabilities and frauds, such as *Multiple Send* [59] and *Honeypots* [60], by replacing the Ponzi-specific features with other suspicious behaviors. Second, our approach of intuitively representing the code execution process as semantic action sequences can assist in broader scenarios involving software code understanding, such as software testing [20]. For example, software engineers typically review source code and develop test cases for deeper analysis. *PonziLens+* can expedite this process by providing both an overview and detailed action patterns during software execution.

Limitations. As smart contract applications become more extensive and complex, often involving multiple contracts, massive potential execution paths can lead to scalability issues. Our path grouping and merging strategies help mitigate such scalability issues to some extent, and further improvements can be made by grouping paths based on functions and filtering actions by their semantics. Also, *PonziLens+* targets Ponzi schemes that embed fraudulent logic within smart contract codes, which is common in blockchain-based scams. However, there are also scams where the fraudulent logic occurs off-chain, such as PlusToken [61], which can only be identified through fund flow analysis and is beyond the scope of *PonziLens+*. Further, symbolic execution's intrinsic limitations may limit the performance of *PonziLens+*, such as ignoring the gas limits causes exploring paths that are not executable in practice or skipping some paths that the Z3 solver cannot resolve. To mitigate these limitations, we can incorporate gas estimation and employ testing techniques with concrete inputs rather than symbolic ones, such as fuzz testing [9].

X. Conclusion

In this work, we first proposed a framework to extract semantically meaningful action sequences to intuitively represent each potential execution path in a smart contract. Then, we proposed *PonziLens+*, a visual analytic system for identifying Ponzi schemes based on these semantic actions, which incorporates three visualization modules to demonstrate the action patterns of a smart contract at three different levels and highlight the features related to Ponzi schemes. We conducted two case studies and in-depth user interviews with 12 users. The results demonstrate that *PonziLens+* is useful and effective in assisting users to easily identify smart Ponzi schemes.

In future work, we will enable users to define custom action patterns beyond Ponzi features, expanding *PonziLens+* to broader smart contract testing and auditing. Furthermore, given that there is an increasing number of cryptocurrency investors using mobile devices (e.g., smartphones) to conduct their investments [62], it is also worth further exploring how *PonziLens+* can be extended to mobile devices.

Acknowledgments

This project is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Proposal ID: T2EP20222-0049).

References

[1] M. Artzrouni, "The mathematics of ponzi schemes," *Mathematical Social Sciences*, vol. 58, no. 2, pp. 190–201, 2009.

[2] M. Bartoletti, S. Carta, T. Cimoli, and R. Saia, "Dissecting ponzi schemes on ethereum: Identification, analysis, and impact," *Future Generation Computer Systems*, vol. 102, pp. 259–277, 2020.

[3] S. Mukherjee, C. Larkin, and S. Corbet, "Cryptocurrency ponzi schemes," *Understanding cryptocurrency fraud: The challenges and headwinds to regulate digital currencies*, vol. 2, p. 111, 2021.

[4] W. Chen, Z. Zheng, J. Cui, E. Ngai, P. Zheng, and Y. Zhou, "Detecting ponzi schemes on ethereum: Towards healthier blockchain technology," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 1409–1418.

[5] "SEC Moves Against $300 Million Crypto Ponzi Scheme," https://www.coindesk.com/policy/2023/02/22/forsage-founders-indicted-for-340m-ponzi-scheme-masquerading-as-defi-platform/, 2022, [Online; Accessed 30-January-2024].

[6] S. Fan, S. Fu, H. Xu, and X. Cheng, "Al-spsd: Anti-leakage smart ponzi schemes detection in blockchain," *Information Processing & Management*, vol. 58, no. 4, p. 102587, 2021.

[7] X. Feng, Q. Shi, X. Li, H. Liu, and L. Wang, "Idponzi: An interpretable detection model for identifying smart ponzi schemes," *Engineering Applications of Artificial Intelligence*, vol. 136, p. 108868, 2024.

[8] "Ponzitracker," https://www.ponzitracker.com/2022-ponzi-schemes, 2022, [Online; Accessed 30-January-2024].

[9] R. Liang, J. Chen, K. He, Y. Wu, G. Deng, R. Du, and C. Wu, "Ponziguard: Detecting ponzi schemes on ethereum with contract runtime behavior graph," in *Proceedings of 2024 IEEE/ACM 46th International Conference on Software Engineering*. IEEE, 2024, pp. 755–766.

[10] Z. Zheng, W. Chen, Z. Zhong, Z. Chen, and Y. Lu, "Securing the ethereum from smart ponzi schemes: Identification using static features," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 5, pp. 1–28, 2023.

[11] W. Chen, Z. Zheng, E. C.-H. Ngai, P. Zheng, and Y. Zhou, "Exploiting blockchain data to detect smart ponzi schemes on ethereum," *IEEE Access*, vol. 7, pp. 37 575–37 586, 2019.

[12] L. Wang, H. Cheng, Z. Zheng, A. Yang, and X. Zhu, "Ponzi scheme detection via oversampling-based long short-term memory for smart contracts," *Knowledge-Based Systems*, vol. 228, p. 107312, 2021.

[13] Y. Liang, W. Wu, K. Lei, and F. Wang, "Data-driven smart ponzi scheme detection," *arXiv preprint arXiv:2108.09305*, 2021.

[14] L. Galletta and F. Pinelli, "Sharpening ponzi schemes detection on ethereum with machine learning," *arXiv preprint arXiv:2301.04872*, 2023.

[15] W. Chen, X. Li, Y. Sui, N. He, H. Wang, L. Wu, and X. Luo, "Sadponzi: Detecting and characterizing ponzi schemes in ethereum smart contracts," *in Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 5, no. 2, pp. 1–30, 2021.

[16] P. Lu, L. Cai, and K. Yin, "Sourcep: Detecting ponzi schemes on ethereum with source code," in *Proceedings of 2024 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2024, pp. 4465–4469.

[17] X. Wen, K. S. Yeo, Y. Wang, L. Cheng, F. Zhu, and M. Zhu, "Code will tell: Visual identification of ponzi schemes on ethereum," in *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–6.

[18] X. Li, T. Chen, X. Luo, T. Zhang, L. Yu, and Z. Xu, "Stan: Towards describing bytecodes of smart contract," in *Proceedings of 2020 IEEE 20th International Conference on Software Quality, Reliability and Security*. IEEE, 2020, pp. 273–284.

[19] G. A. Pierro, "Smart-graph: Graphical representations for smart contract on the ethereum blockchain," in *Proceedings of 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering*. IEEE, 2021, pp. 708–714.

[20] C. Zhou, M. Wang, J. Liang, Z. Liu, C. Sun, and Y. Jiang, "Visfuzz: Understanding and intervening fuzzing with interactive visualization," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 1078–1081.

[21] Y. Lou, Y. Zhang, and S. Chen, "Ponzi contracts detection based on improved convolutional neural network," in *Proceedings of 2020 IEEE International Conference on Services Computing*. IEEE, 2020, pp. 353–360.

[22] G. Ibba, G. A. Pierro, and M. Di Francesco, "Evaluating machine-learning techniques for detecting smart ponzi schemes," in *Proceedings of 2021 IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain*. IEEE, 2021, pp. 34–40.

[23] S. Fan, S. Fu, H. Xu, and C. Zhu, "Expose your mask: Smart ponzi schemes detection on blockchain," in *Proceedings of 2020 International Joint Conference on Neural Networks*, 2020, pp. 1–7.

[24] W. Sun, G. Xu, Z. Yang, and Z. Chen, "Early detection of smart ponzi scheme contracts based on behavior forest similarity," in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security*. IEEE, 2020, pp. 297–309.

[25] N. Tovanich, N. Heulot, J.-D. Fekete, and P. Isenberg, "Visualization of blockchain data: a systematic review," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 7, pp. 3135–3152, 2019.

[26] G. Di Battista, V. Di Donato, M. Patrignani, M. Pizzonia, V. Roselli, and R. Tamassia, "Bitconeview: Visualization of flows in the bitcoin transaction graph," in *Proceedings of 2015 IEEE Symposium on Visualization for Cyber Security*. IEEE, 2015, pp. 1–8.

[27] M. Ahmed, I. Shumailov, and R. Anderson, "Tendrils of crime: Visualizing the diffusion of stolen bitcoins," in *Proceedings of Graphical Models for Security: 5th International Workshop*. Springer, 2019, pp. 1–12.

[28] S. Bistarelli and F. Santini, "Go with the-bitcoin-flow, with visual analytics," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, 2017, pp. 1–6.

[29] D. McGinn, D. Birch, D. Akroyd, M. Molina-Solana, Y. Guo, and W. J. Knottenbelt, "Visualizing dynamic bitcoin transaction patterns," *IEEE Transactions on Big Data*, vol. 4, no. 2, pp. 109–119, 2016.

[30] X. Wen, Y. Wang, X. Yue, F. Zhu, and M. Zhu, "Nftdisk: Visual detection of wash trading in nft markets," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–15.

[31] P. Isenberg, C. Kinkeldey, and J.-D. Fekete, "Exploring entity behavior on the bitcoin blockchain," in *Proceedings of VIS 2017-IEEE Conference on Visualization*, 2017, pp. 1–2.

[32] C. Kinkeldey, J.-D. Fekete, and P. Isenberg, "Bitconduite: Visualizing and analyzing activity on the bitcoin network," in *Posters of EuroVis 2017-Eurographics Conference on Visualization*, 2017, p. 3.

[33] X. Yue, X. Shu, X. Zhu, X. Du, Z. Yu, D. Papadopoulos, and S. Liu, "Bitextract: Interactive visualization for extracting bitcoin exchange intelligence," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 162–171, 2018.

[34] R. Norvill, B. B. F. Pontiveros, R. State, and A. Cullen, "Visual emulation for ethereum's virtual machine," in *Proceedings of 2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–4.

[35] S. Devkota and K. E. Isaacs, "Cfgexplorer: Designing a visual control flow analytics system around basic program analysis operations," in *Computer Graphics Forum*, vol. 37, no. 3. Wiley Online Library, 2018, pp. 453–464.

[36] S. Devkota, M. Legendre, A. Kunen, P. Aschwanden, and K. E. Isaacs, "Cfgconf: Supporting high level requirements for visualizing control flow graphs," *arXiv e-prints*, p. arXiv: 2108.03047, 2021.

[37] X. Zhou, Y. Chen, H. Guo, X. Chen, and Y. Huang, "Security code recommendations for smart contract," in *Proceedings of 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering*, 2023, pp. 190–200.

[38] S. Diehl, *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer Science & Business Media, 2007.

[39] N. Chotisarn, L. Merino, X. Zheng, S. Lonapalawong, T. Zhang, M. Xu, and W. Chen, "A systematic literature review of modern software visualization," *Journal of Visualization*, vol. 23, pp. 539–558, 2020.

[40] D. Hayatpur, D. Wigdor, and H. Xia, "Crosscode: Multi-level visualization of program execution," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–13.

[41] K. E. Isaacs and T. Gamblin, "Preserving command line workflow for a package management system using ascii dag visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 9, pp. 2804–2820, 2018.

[42] Y. Yoon, B. A. Myers, and S. Koo, "Visualization of fine-grained code change history," in *2013 IEEE symposium on visual languages and human centric computing*. IEEE, 2013, pp. 119–126.

[43] Y. Kim, J. Kim, H. Jeon, Y.-H. Kim, H. Song, B. Kim, and J. Seo, "Githru: visual analytics for understanding software development history through git metadata analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 656–666, 2020.

[44] F. Zhou, Y. Fan, S. Lv, L. Jiang, Z. Chen, J. Yuan, F. Han, H. Jiang, G. Bai, and Y. Zhao, "Fctree: Visualization of function calls in execution," *Information and Software Technology*, p. 107545, 2024.

[45] K. E. Isaacs, P.-T. Bremer, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, and B. Hamann, "Combing the communication hairball: Visualizing parallel execution traces using logical time," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2349–2358, 2014.

[46] K. Xu, Y. Wang, L. Yang, Y. Wang, B. Qiao, S. Qin, Y. Xu, H. Zhang, and H. Qu, "Clouddet: Interactive visual analysis of anomalous performances in cloud computing systems," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 1107–1117, 2019.

[47] Y. Sun, Y. Zhang, A. Mosallaei, M. D. Shah, C. Dunne, and D. Kaeli, "Daisen: A framework for visualizing detailed gpu execution," in *Com-*

puter Graphics Forum, vol. 40, no. 3. Wiley Online Library, 2021, pp. 239–250.

[48] H. Guo, S. Di, R. Gupta, T. Peterka, and F. Cappello, "La valse: Scalable log visualization for fault characterization in supercomputers." in *Proceedings of EuroVis*, 2018, pp. 91–100.

[49] L. El Hassouni and A. Ouchekkir, "Smart contracts: An emerging business model in decentralized finance," in *International Conference on Digital Technologies and Applications*. Springer, 2023, pp. 197–207.

[50] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[51] M. Soud, G. Hjálmtỳsson, and M. Hamdaqa, "Dissecting smart contract languages: A survey," *arXiv preprint arXiv:2310.02799*, 2023.

[52] "Opcodes for the evm," https://ethereum.org/en/developers/docs/evm/opcodes/, 2024, [Online; Accessed 6-November-2024].

[53] F. Contro, M. Crosara, M. Ceccato, and M. Dalla Preda, "Ethersolve: Computing an accurate control-flow graph from ethereum bytecode," in *Proceedings of 2021 IEEE/ACM 29th International Conference on Program Comprehension*. IEEE, 2021, pp. 127–137.

[54] T. Moore, J. Han, and R. Clayton, "The postmodern ponzi scheme: Empirical analysis of high-yield investment programs," in *Proceedings of Financial Cryptography and Data Security*, vol. 7397. Springer, 2012, pp. 41–56.

[55] J. Krupp and C. Rossow, "Teether: Gnawing at ethereum to automatically exploit smart contracts," in *Proceedings of 27th USENIX Security Symposium*, 2018, pp. 1317–1333.

[56] J. He, M. Balunović, N. Ambroladze, P. Tsankov, and M. Vechev, "Learning to fuzz from symbolic execution with application to smart contracts," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 531–548.

[57] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[58] R. Kosara, F. Bendix, and H. Hauser, "Parallel sets: Interactive exploration and visual analysis of categorical data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 4, pp. 558–568, 2006.

[59] "Dos with failed call," https://swcregistry.io/docs/SWC-113/, 2020, [Online; Accessed 27-August-2024].

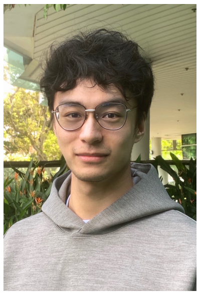
[60] C. F. Torres, M. Steichen *et al.*, "The art of the scam: Demystifying honeypots in ethereum smart contracts," in *Proceedings of 28th USENIX Security Symposium*, 2019, pp. 1591–1607.

[61] S. Zhang, D. Zhang, J. Zheng, W. Aerts, and D. Xu, "Plus token and investor searching behaviour–a cryptocurrency ponzi scheme," *Accounting & Finance*, vol. 63, no. 4, pp. 4713–4728, 2023.

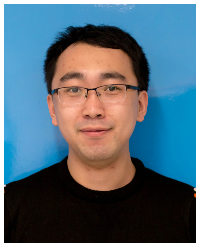
[62] M. M. Mirza, A. Ozer, and U. Karabiyik, "Mobile cyber forensic investigations of web3 wallets on android and ios," *Applied Sciences*, vol. 12, no. 21, p. 11180, 2022.

Xiaolin Wen is currently a Ph.D student in the College of Computing and Data Science at Nanyang Technological University (NTU). His research interests mainly focus on visualization and human-computer interaction techniques in Fin-tech and Web3. He received his master's degree in Computer Science and Technology from Sichuan University in 2023 and his dual bachelor's degree in Computer Science and Financial Engineering from Sichuan University in 2016. For more information, kindly visit https://wenxiaolin.com/.

Tai D. Nguyen received his Ph.D. degree in computer science from the School of Computing and Information Systems at Singapore Management University (SMU). He is currently a research fellow at SMU. His current research interests include formal methods, smart contract security, and AI security. For more information, kindly visit https://duytai.github.io/.



Shaolun Ruan is currently a Ph.D. candidate in School of Computing and Information Systems at Singapore Management University (SMU). His work focuses on developing novel graphical representations that enable a more effective and smoother analysis for humans using machines, leveraging the methods from Data Visualization and Human-computer Interaction. He received his bachelor's degree from the University of Electronic Science and Technology of China (UESTC) in 2019. For more information, kindly visit https://shaolun-ruan.com/.



Qiaomu Shen received the PhD degree in computer science from The Hong Kong University of Science and Technology in 2020. He is currently an research assistant professor in Southern University of Science and Technology(SUSTECH). Before joining SUSTECH, he worked as a senior developer at Noah's Ark Lab for Huawei Technologies. His current research interests include spatial-temporal visualization, urban computing, and visual analytics of complex system.

Feida Zhu is currently a tenured Associate Professor and Associate Dean at the School of Computing and Information Systems, Singapore Management University. He obtained his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign (UIUC) in 2009. His research interests include AI and collaborative intelligence, blockchain, data asset and AI governance, with emphasis on their application to business, financial and consumer innovation. Prof. ZHU has over 100 peer-reviewed research publications at top international venues, including ICDE, VLDB, SIGMOD, KDD, WWW, JMLR, TODS, TKDE, etc.

Jun Sun is currently a professor at Singapore Management University (SMU). He received Bachelor and PhD degrees in computing science from National University of Singapore (NUS) in 2002 and 2006. He has been a faculty member since 2010. He has received the prestigious LEE KUAN YEW fellowship twice in 2007 and 2022. Jun's research interests include formal methods, program analysis and lately AI security. His profile can be found at https://sunjun.site.

Yong Wang is currently an assistant professor in the College of Computing and Data Science, Nanyang Technological University. Before that, he worked as an assistant professor at Singapore Management University from 2020 to 2024. His research interests include data visualization, HCI and human-AI collaboration, with an emphasis on their application to FinTech, quantum computing and online learning. He obtained his Ph.D. in Computer Science from Hong Kong University of Science and Technology. He received his B.E. and M.E. from Harbin Institute of Technology and Huazhong University of Science and Technology, respectively. For more details, please refer to http://yong-wang.org.

Appendix

A. Path Merging Strategy

In the appendix, we describe the details of our **Path Merging Strategy** used in Path Grouping Module as shown in Algorithm 1. Our path-merging strategy has two steps: **Step 1: Merge paths sharing the same basic block sub-sequence.** Represent each execution path by its basic block number sequence, group paths with identical sub-sequences and no order conflicts, and merge them to create the longest sequence that includes all basic blocks of these paths. **Step 2: Separate paths with different actions in merged basic blocks**. Compare the semantic action sequences of different paths in the merged basic blocks from Step 1. If different actions are found within the same basic block, separate them to maintain individual actions. After the above two steps, we visualize the merged execution paths in the Path Group Module.

Algorithm 1: Path Merging Strategy

```
Input: $L$: Execution paths list, where each path consists of a sequence of basic blocks, and each basic block contains a
    sequence of actions.
Output: $MergedPaths$: List of merged paths, where the same basic blocks are merged and the different actions in
    merged blocks are separated.
/* Step1: Merge paths sharing the same basic block sub-sequence */
/* Map each path in $L$ into a sequence of basic block index */
$BasicBlockSequences \leftarrow MapToBasicBlockSequences(L)$;
/* Divide basic block sequences into sub-groups by the rule: sharing the same
   basic block sub-sequence without order conflicts */
$BasicBlockSequencesSubGroups \leftarrow GroupByRule(BasicBlockSequences)$;
$FullSequenceList \leftarrow []$;        /* List of full sequence for each sub-group */
$MergedPaths \leftarrow []$;        /* List of merged paths with different actions separated */
/* For each sub-group of paths that can be merged */
for each $subGroup$ in $BasicBlockSequencesSubGroups$ do
    /* Merge all basic block sequences of a sub-group into the longest full
       sequence */
    $FullSequence \leftarrow MergeIntoFullSequence(subGroup)$;
    $FullSequenceList \leftarrow Add(FullSequence)$;
    /* Step2: Separate paths with different actions in merged basic blocks */
    $Differences \leftarrow []$;        /* Different actions in the merged paths */
    for each $BasicBlock$ in $FullSequence$ do
        /* Get a set of paths containing this basic block from the sub-group */
        $PathsSet \leftarrow GetPathByBlock(BasicBlock, subGroup)$;
        /* Fetch action sequences of this basic block from the original path data */
        $ActionSequences \leftarrow GetActionSequences(PathsSet, L)$;
        $Difference \leftarrow CompareActions(ActionSequences)$;
        /* Compare the differences in actions among paths */
        $Differences \leftarrow Add(Difference)$
    end
    $MergedPaths \leftarrow Add(FullSequence, Differences)$
end
Return $MergedPaths$
```

B. Participant Information Table

In the user interviews, we gathered participant profiles, including gender, age, and experience with web3 applications and smart contract auditing. We also asked them to describe how smart contracts relate to their daily work. All the participants have enough background in blockchain and smart contracts, and they have experience in investing in smart contracts. For participants who preferred not to describe their specific jobs, we just marked them as "A web3 investor who has invested in smart contracts" in this table.

TABLE I
The detailed information of the user interview participants. All participants are experienced in smart contract investment. U1-U6 have experience in auditing smart contracts, while U7-U12 lack experience in smart contracts auditing.

ID	Gender	Age	Web3 Experience	Auditing Experience	Description
U1	Male	31	60 months	48 months	A PhD expertise in smart contract security, working at a web3 security company.
U2	Male	27	36 months	24 months	A PhD candidate with a published paper on smart Ponzi scheme detection.
U3	Male	29	42 months	6 months	A PhD candidate in blockchain analysis, interning at a web3 security company.
U4	Male	35	30 months	24 months	A smart contract auditor at a web3 security company.
U5	Male	34	18 months	8 months	A team leader of web3 projects at an internet company.
U6	Male	23	6 months	6 months	A master student working on smart Ponzi scheme detection.
U7	Male	32	48 months	0 months	A creator of a web3 community and a key opinion leader on Twitter.
U8	Male	25	36 months	0 months	A PhD candidate expertise in Cryptography.
U9	Male	30	24 months	0 months	A web3 investor who has invested in smart contracts.
U10	Female	27	18 months	0 months	A web3 investor who has invested in smart contracts.
U11	Female	22	12 months	0 months	An operations manager at a web3 security company.
U12	Male	25	10 months	0 months	A web3 investor who has invested in smart contracts.