IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 20 February 2024, accepted 3 April 2024, date of publication 9 April 2024, date of current version 16 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3386751

RESEARCH ARTICLE

An Annotation Assisted Smart Contracts Generation Method

YONG CHEN, DEFENG HU, CHAO XU, AND NANNAN CHEN
School of Computer Science, Nanjing Audit University, Nanjing, Jiangsu 211815, China

Corresponding authors: Defeng Hu (17751771036@163.com), Yong Chen (chenyong@nau.edu.cn), and Chao Xu (xuchao@nau.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 71972102, Grant 71972102, and Grant 61902189; in part by the Universities Natural Science Research Project of Jiangsu Province under Grant 20KJB520026 and Grant 20KJA520002; and in part by the Foundation for Young Teachers of Nanjing Audit University under Grant 19QNPY018.

ABSTRACT Smart contracts are rapidly applied in many fields, with their varied types and increasing complexity. A sharp increase in the method development demands seems to be certain. However, this type of development has its unique programming language and security requirements, making it difficult for regular software personnel to adapt quickly. It is important to realize that the development efficiency is application-specific and that getting this application issue solved is critical for its further development. To this end, we propose a new, automatic, and intelligent contract-generation method, based on code annotation. First of all, combined with the semantic annotation information of smart contract code association, a clustering analysis model is built to realize fast and accurate clustering with functions similar to smart contracts. Then, based on the Char-RNN network, a multi-level and automatic generation method of intelligent contract knowledge base is built to realize the automatic generation at different levels, such as the contract layer, function layer, and interface layer. Finally, by using text matching technology and by calculating the semantic similarity of the user text demands as well as the smart contract knowledge base annotation, the relevant contract code is automatically extracted for users to choose, with the aims of improving the method efficiency and to meet the needs of different users. To test the effectiveness of the method, with the aid of bilingual quality assessment BLEU and Mythril, VaaS, and other code security tools for evaluation are used and results are compared with the existing method. The generated code BLEU average score was increased by 27% and the average accuracy was increased by 11.5%. Therefore, the smart contract generated by our method is relatively accurate and reliable.

INDEX TERMS Smart contract, annotation, char-RNN, multi-level code generation.

I. INTRODUCTION

Smart contracts play a crucial role in blockchain technology, as they allow for transactions to be completed without relying on intermediaries. This break from traditional transaction modes has wide-ranging applications in various fields, including education, voting, real estate, entertainment, the Internet of Things, supply chain management, healthcare, and more [35]. According to [29] and [35], smart contracts are executable programs deployed on a blockchain to facilitate, validate, and execute transactions. In essence, a blockchain is a distributed database that records transactions of smart contract fragments, which can be interacted with by untrusted nodes. The application scenarios of blockchain platforms significantly extended by separating transactions from external trusted institutions. Currently, more and more blockchain platforms support smart contracts, such as Ethereum, Hyperledger Fabric, and VNT Chain, all of which provide programming languages for smart contract development. Among these, Solidity is the most popular programming language used on the Ethereum platform [29].

In the past decade, smart contracts have shown great potential for a wide range of applications. However, their security remains a crucial issue hindering their development and causing substantial financial losses [30]. Some of the most notable incidents include the DAO security breach

The associate editor coordinating the review of this manuscript and approving it for publication was Chuan Li.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

in June 2016, which resulted in a loss of $50 million, the Parity multi-signature wallet security breach in July 2017, which resulted in a loss of almost $200 million, and the BEC token theft in April 2018, which resulted in a loss of $900 million due to a single line of code. These security incidents have caught the attention of numerous researchers. Scholars such as [10] and [22] have highlighted that smart contract security mainly involves contract security and privacy security issues. Contract security primarily pertains to the design, deployment, maintenance, and testing stages of smart contracts. It necessitates the development of smart contracts without code vulnerabilities or design defects, and the enhancement of smart contract vulnerability detection tools. Additionally, the immutability of smart contracts contributes to their credibility, as they cannot be modified once deployed online. However, if there are loopholes, correcting them can be difficult, and unreliable smart contracts can severely compromise the privacy of the contract code and related data.

With the rapid expansion of smart contracts, their applications are becoming increasingly diverse. In addition to ensuring the security of the contracts themselves, designing smart contracts requires knowledge and skills in the relevant application areas. Consequently, there is often a disconnect between contract developers and designers in the smart contract field. While contract developers may be skilled in smart contract programming languages, they may lack an understanding of contract logic. Conversely, designers may be familiar with business rules but lack a comprehension of smart contracts. This results in lower efficiency of smart contract development and hinders the pace of their progress. Furthermore, a survey cited in [27] reveals that as of October 2022, the number of smart contracts deployed and released in Ethereum has reached 51 million, and the market size is expected to grow at an annual rate of 32%, involving billions of dollars. Due to the enormous market size, the efficiency and security requirements for smart contract development have become more stringent. Therefore, developing accurate and secure smart contracts rapidly is imperative to meet the evolving needs of diverse users.

User-friendliness refers to the design and development of smart contracts, taking into account the needs, capabilities, and habits of many smart contract developers and the reusability of a large number of codes, to provide a simple, intuitive, and easy-to-operate interface, to promote the efficiency of developers' code development and reduce the difficulty of development. At present, most smart contract researches mainly focus on security detection, while ignoring the development efficiency and user-friendliness of smart contract programming. The interaction mode of the smart contract development environment is still based on pure code writing, and there is a lack of additional auxiliary information to assist developers in code design. Therefore, the professional requirements for users to master relevant programming languages are relatively high. This tends to reduce the friendliness and efficiency of the developer's interaction with the code to some extent. On the one hand, non-friendliness is mainly reflected in the following two problems: First, many non-professional developers (such as in the education field) do not know much about the language of the smart contract field, and it is often difficult to develop target contracts according to their needs. Secondly, with the rapid development of smart contracts, a large number of basic contract code is often reused in the development of a high frequency, and developers are still completing my writing, which undoubtedly greatly increases the redundancy of development code. On the other hand, although large models such as ChatGPT have very good code generation capabilities, the development work for smart contracts still has problems such as poor quality and inaccurate code generation, lack of semantic information, and can only generate basic contract code. The limitations of the current work provide a certain motivation for this study. We conduct cluster analysis through a large number of smart contract data and extract smart contract code with a high usage rate as the data set for model training, to reduce the extra cost of basic code development for developers and design relevant development interfaces for developers to use. Using code comment information to help developers match and generate code, and improve User-friendliness and efficiency. Although some research [16] to improve the efficiency of the smart development contract, but still need to improve the ability of developers to quickly understand intelligent contract code, can be further optimized code development efficiency and accuracy. Therefore, we propose an automatic smart contract generation method based on annotation assistance. The main contributions of this paper are as follows:

1) Designed a smart contract similarity assessment based on annotation-assisted cluster analysis. The smart contracts obtained by the crawler program were preprocessed and divided into code hierarchies. Annotation information and code were given different weights to perform clustering analysis on the source code of smart contracts at various levels, resulting in obtaining hierarchical smart contract source code documents with similar features and ensuring the accuracy of the smart contract generation model.
2) A multi-level smart contract code auto-generation model based on deep learning was constructed. Building upon the Char-run model in deep learning and using a bidirectional GRU (directional GRU) [14], [24] as the hidden layer to replace the basic RNN model, the model takes the smart contract source code at various levels as input to generate unified, semantically hierarchical code.
3) An annotation-assisted method for adaptive matching of smart contracts was designed. Deep learning similarity algorithms were employed, with the Jaccard coefficient serving as the calculation index. Additionally, a user-friendly interactive interface was created. By calculating the similarity between the user's input requirements and the corresponding annotation

information of the code, adaptive matching of smart contract code with the assistance of annotation information was achieved.

II. RELATED WORK

Currently, research on smart contract security issues mainly focuses on the detection of smart contract code vulnerabilities, which is a highly concerning issue in the blockchain field. References [20], [22], and [27] analyzed smart contract vulnerabilities from three perspectives: the Solidity code layer, the EVM execution layer, and the blockchain system layer, where the Solidity code layer includes over ten vulnerabilities such as re-entry vulnerabilities (The Dao attack) and integer overflow vulnerabilities, the EVM execution layer involves vulnerabilities such as short address vulnerabilities and ether loss vulnerabilities, and the blockchain system layer mainly include three vulnerabilities such as timestamp dependence vulnerabilities. Smart contract security vulnerabilities not only harm users' interests but also undermine the credibility system of the blockchain system. To prevent vulnerabilities from being exploited by criminals, researchers have begun to try various methods to prevent the occurrence of vulnerabilities. In recent years, the detection methods for smart contract vulnerabilities [20] mainly include formal verification, symbolic execution, fuzz testing, intermediate representation, and deep learning methods. Therefore, tools such as VaaS, Oyente [16], Mythril [16] and SmartCheck [8] have been widely used in the detection of smart contract vulnerabilities, and the security issues of smart contracts have been effectively improved. In addition, in terms of friendly interaction between smart contracts and developers, [8], [16] proposed a method for generating specific domain smart contracts automatically. They constructed a smart contract automatic generation model based on long short-term memory recurrent neural networks (LSTM) in deep learning, used SmartCheck to detect generated code, and obtained good experimental results. Finally, by designing a well-designed interactive interface, the friendliness and efficiency of programming were improved.

Code annotations [21] as auxiliary information of source code, mark the implementation content and purpose of the code, which helps developers understand, develop, and maintain source code. In recent years, researchers have paid more and more attention to various aspects of code annotations, mainly in the direction of code annotation generation and quality evaluation. Reference [5] conducted specific research on automatic code annotation generation, which is currently mainly divided into template-based generation methods, information retrieval-based generation methods, and deep learning-based generation methods. The core ideas of these three methods were analyzed in-depth, and experimental results were compared and analyzed, indicating that code annotation generation has rich research value and application prospects. In addition, code annotation quality evaluation [5], [9], [21], [23] is one of its research focuses, and high-quality code annotations effectively reduce the cost of developers' understanding of the code. Especially when developers face a large workload of code, due to lack of experience, lack of professional domain knowledge, or careless mistakes, code errors or omissions often occur, and reasonable code annotations can greatly alleviate this problem. In view of the current research on code annotations, they play a crucial role in the semanticization of code and in enhancing developers' understanding of the code.

In research on code generation using deep learning [3], [11] pointed out that recursive neural networks (RNN), convolutional neural networks (CNN), and generative adversarial networks (GAN) are the main research directions. The idea is to use various neural network structures to build code generation models and generate corresponding code by training the models with large amounts of data. Unlike traditional methods, long short-term memory recursive networks (LSTM) [13], gated recurrent units [13] are applied to various basic neural network models, and selective retention of sequence information effectively improves the accuracy and flexibility of code. In recent years, with the continuous development of deep learning, numerous studies on code generation are being continuously optimized, thus code generation techniques based on deep learning have good prospects for application.

The fairness and decentralization of smart contracts greatly promote the wide application of smart contracts in various fields, and many researchers have also realized the research on the application of smart contracts. Reference [31] and other experts have integrated smart contract and blockchain technology into the Internet of Things, and thus innovatively proposed an access control system based on smart contract tokens. The system has added the smart contract system to supervise and manage all kinds of access control events, greatly improving security and scalability. Decentralized applications are a common scenario for smart contracts, and [4] proposes an effective penetration framework to address the security concerns of such scenarios. In addition, smart contracts also have important applications in other fields. For example, [1] and others combined smart contracts with a college education to create an intelligent, safe, and fair learning environment for students. [25] Siddiqui et al. have applied smart contracts in various fields such as smart city management and real estate development. Smart contracts, as the contract layer in the blockchain, greatly change and simplify the way many business and social interactions, driving the development of the digital economy and decentralized applications.

We have summarized and summarized the research results in related work, and drawn relevant tables to show them, as shown in Table 1.In summary, based on existing research and relevant content, the proposed smart contract generation technology based on code comments and deep learning techniques in this paper has good feasibility and rationality. Moreover, the paper has certain innovations because the research on code generation often focuses only on the

code itself, and neglects the importance of code comment information.

III. SMART CONTRACT AUTOMATIC GENERATION AND MATCHING METHOD ASSISTED BY ANNOTATION INFORMATION

A. THE OVERALL FRAMEWORK OF THE APPROACH

The research framework of this article can be divided into three parts: first, using a web crawler program to crawl the source code of smart contracts from Ethereum, dividing them into levels based on their code and annotation documents, and establishing a corresponding mapping relationship through file numbers. In the annotation-assisted smart contract source code clustering stage, we preprocess the clustered code and annotation documents separately as input, calculate their cosine distance values from two dimensions of annotations and code, and use the K-means algorithm to generate clusters of code in each level, thereby extracting hierarchical code with high similarity. Secondly, in the smart contract code generation part, to ensure the accuracy of code generations, we preprocess the hierarchical code after clustering twice and use a recurrent neural model as the smart contract code auto-generation model to generate unified basic code. The generated code and annotations are stored in the database, and developers can match the code in the database by inputting their own demand information. Finally, we designed a convenient human-computer interaction interface, which calculates the text similarity between the demand information entered by the user in the smart contract interaction interface and the corresponding annotation content of the code, thus achieving annotation-assisted adaptive matching of smart contracts. The specific overall frame diagram is Fig 1:

B. ANNOTATION-GUIDED SMART CONTRACT CLUSTERING

In terms of the research content of code analysis, most studies are limited to the code content itself, while often ignoring the information brought by annotations for the corresponding code. Therefore, in addition to the basic dimension of the code itself, annotations are another focus of this research. We will use clustering in two dimensions of annotation text and code to extract similar hierarchical smart contract code. The specific steps of clustering are as follows:

1) THE SELECTION OF THE OPTIMAL CLUSTERING CENTER POINT K

Before the K-means clustering algorithm is carried out, the value of the number of cluster centers k must be determined first. The number of smart contracts used in this paper is large and the real number of clusters cannot be determined. Secondly, smart contract samples can be regarded as data in the form of a simple text structure. The elbow method is a method to determine the optimal cluster number by observing the relationship between the Sum of Squared Errors (*SSE*) within the cluster and the cluster number K. In the Kmeans clustering algorithm, *SSE* is an index that measures the sum of squares of the distance between data points in a cluster and its cluster center. In the clustering process, with the increase of cluster number k, sample division will be more refined, and the degree of aggregation of each cluster will gradually increase, so the square error and *SSE* will naturally become smaller. When k is less than the real cluster number,*SSE* will decrease greatly because the increase of k will greatly increase the degree of aggregation of each cluster. However, when k reaches the real cluster number, the return on the degree of aggregation obtained by increasing k will rapidly decrease, so *SSE* will decrease sharply, and then tend to be flat with the continuous increase of k value. That is to say, the relationship between *SSE* and k is the shape of an elbow, and the corresponding k value of this elbow is the real cluster number of data. The elbow method is an unsupervised simple and intuitive method that can help estimate the optimal cluster number K when there is no prior knowledge about the true number of clusters in the data. In the case of relatively simple data structures and cluster shapes, the elbow method is often effective in finding elbows and providing reasonable clustering number recommendations. Based on the experimental data in this paper, we use the elbow method to obtain the K value.

2) CLUSTERING OF SMART CONTRACTS BASED ON ANNOTATION GUIDANCE

The annotation contains more source code description information, which can help developers quickly understand and read the source code of smart contracts. In order to improve the accuracy of the research, we must ensure that the clustered smart contract documents have more annotation content for the acquired smart contract source code documents. Smart contract files with a large amount of text contain rich annotation information, so we reserve smart contract files with a file size between 50KB and 80KB as clustering data sets. Secondly, the smart contract source program document contains many JSON format texts, binary code texts, and smart contract texts without annotations, and these texts should be preprocessed.

The smart contract source code is generally composed of four levels of code bodies: ''function'', ''contract'', ''interface'', and ''library''. For the clustered data sets, we have carried out the above-mentioned hierarchical division, which can help developers quickly generate and manage object code. Secondly, annotation is the information description of the code, and the annotation and source code are separated and stored to effectively avoid the interference of invalid words and messy information in the clustering process of these two types of samples. Then establish the link between the code body and the corresponding annotation document through the document serial number. For example, a function in the ''function'' layer code is stored in the function1 file, and the corresponding annotation document is correspondingly stored in the function1_note file. The smart contract code

TABLE 1. Research summary of related work.

In the relevant work part, the first paragraph is mainly to elaborate on the security issues of the current smart contract, which is also an issue that blockchain attaches great importance to. At present, the security detection tools of smart contracts have been very mature, and most relevant studies use this kind of security verification of smart contract code. This provides a certain experimental basis for the subsequent use of safety detection tools in this paper.
The second part is the research on code comments. Researchers attach great importance to the research on comments. At present, the main research directions are comment generation and quality assessment. As annotation information of annotations, code comment contains a large amount of description information about the code. Therefore, most current research directions tend to focus on the single direction of comment or code, ignoring the connection between the two, which provides a very good starting point for this paper. Adding annotation to the dimension of code semantic analysis is the innovation of this paper.
The third part is the study of code generation using deep learning. At present, a large number of researchers have realized the code generation task of deep learning models such as CNN and RNN. Secondly, LSTM and GRU models have also been widely used. Therefore, neural network has a good application basis and prospect in the task of code generation, which provides a certain research basis for this paper.
The fourth part mainly describes the current research results of the wide application of smart contracts in various scenarios, smart contracts in the Internet of things, penetration testing, real estate development, education and other fields play an extremely important role. Many researchers are committed to the research of the combination of smart contracts with other fields and the research of smart contract security testing, so the demand for the high efficiency and friendliness of smart contract development will become a problem that cannot be ignored in the future.

samples and annotation information samples that establish the serial number mapping relationship are the experimental data sets of this research.

K-means is a clustering algorithm based on distance division, which considers that the closer the two targets are, the greater the similarity is. In this study, we have chosen this algorithm as the clustering method and used Cosine Distance as the calculation basis. We calculated the Cosine Distance values of the content of each layer of the smart contract code and the corresponding annotation content separately. Finally, we gave a certain weight to the two cosine values and added them to obtain the final judgment distance of the sample. And by measuring the distance between each sample, we achieved clustering. To improve the accuracy of clustering, preprocessing of the smart contract dataset should be performed before clustering. Firstly, some words in the smart contract code appear many times but cannot reflect their importance in the code, such as ''public'', ''return'', ''internal'', etc., which cannot be used as feature terms. Therefore, we set these words, such as ''public'', as stop words. Similarly, stop words such as ''//'' and ''/**/'' should be set in the annotation document clustering. Secondly, due to the characteristics of the Solidity language, many identifiers contain upper and lower case letters, such as ''Contract'' and variable names. Therefore, we treat upper and lower case letters as consistent.

During the clustering process, each document needs to be vectorized to calculate the distance. We use the spatial vector model (VSM) to consider the code and annotation documents as vectors composed of multiple groups of different feature items and corresponding weights. The formula is as follows:

$$D_i = D_i\left(t_1, w_1; t_2, w_2; \cdots; t_n, w_n\right) \tag{1}$$

$$G_i = G_i\left(t_1, w_1; t_2, w_2; \cdots; t_n, w_n\right) \tag{2}$$

Then D_i and G_i represent the ith code and comment document, t_n is each of these identifiers or keywords, w_n is the corresponding word frequency. In the experiment, multiple cluster centers were selected to calculate the cosine distance value between the smart contract code and the central document at each level. At the same time, the cosine distance between the corresponding annotated document and the central document was calculated, and the weight was given and the proportion is λ: (1)-λ). After the weighted calculation, the formula is as formula 3, shown at the bottom of page 7. The clustering procedure is shown in Algorithm 1, specifically as follows:

1) First, remove invalid documents from the smart contract dataset to avoid interference from non-contract structural code. Additionally, perform stop-word processing on both code and comment documents separately.
2) We vectorize each document using the VSM, randomly select K cluster center points, set the loop starting flag of Algorithm 1 to TRUE, and calculate the cosine distance between all code and corresponding annotation documents and center points using formula 3. Then, we assign them to the center document that is closest in distance (lines 5-13 in Algorithm 1).
3) After the previous step is completed, the average distance of each document in K clusters is recalculated

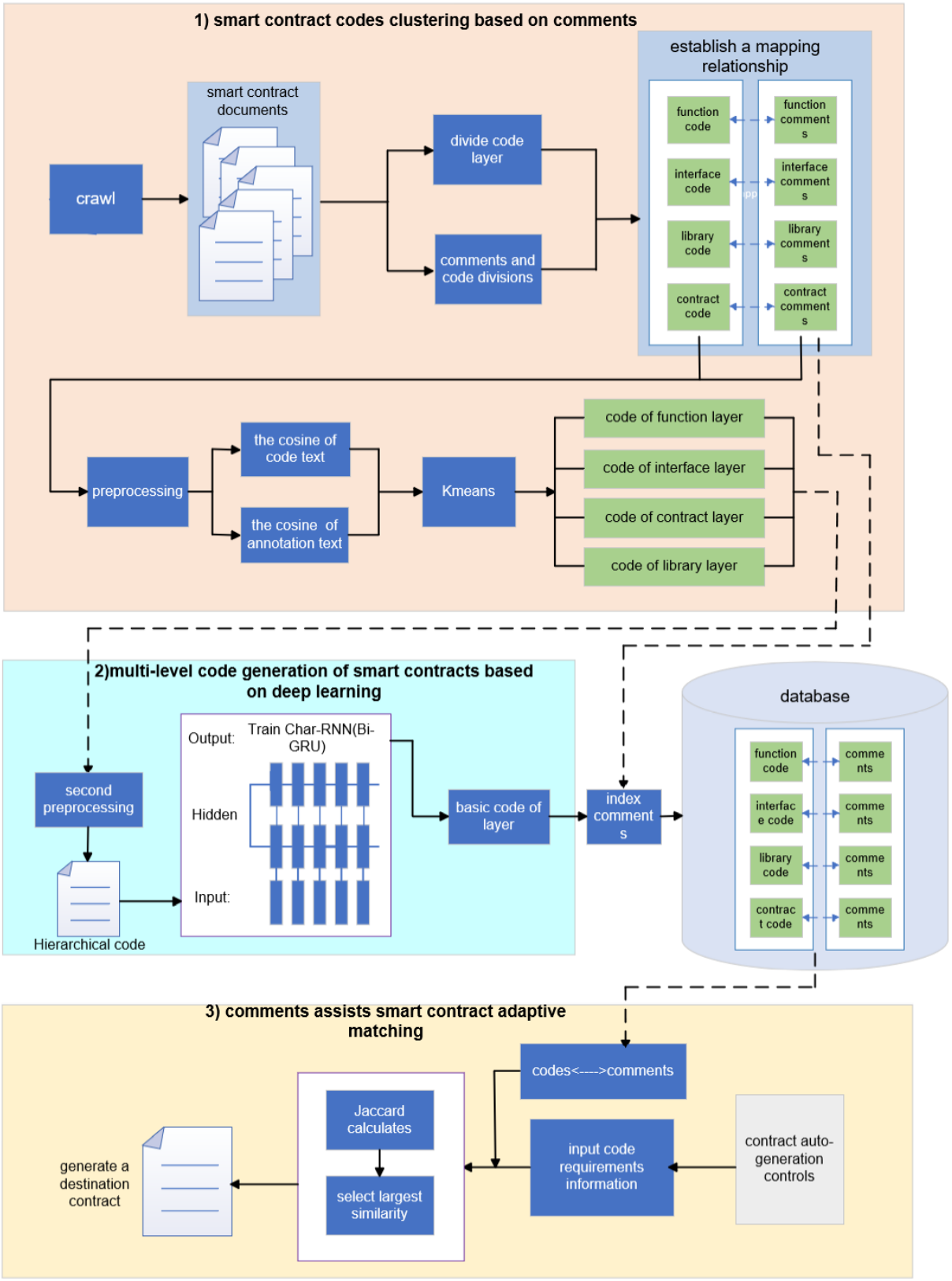

FIGURE 1. Overall framework.

Algorithm 1 Smart Contract Kmeans Clustering Algorithm Based on Annotation Guidance

Require: Hierarchical smart contract source code collection E(1,2,...,i,..,N) and annotation document collection G(1,2,...,i,..,N), and the number of cluster centers K

Ensure: Clusters of smart contract documents after clustering

1: Randomly initialize K cluster centers K (center1, center2,..., cnterK), 0<cnterK<N+1;
2: $Flag \Leftarrow TRUE$
3: **while** $Flag \neq FALSE$ **do**
4: Initialize the data, the number of smart contract source code documents and annotation documents is N, and $s \Leftarrow 0$
5: **while** $s < N$ **do**
6: $cosdistance \Leftarrow cos(i, K(centerK)$
7: **for** $i \Leftarrow 0$ to K **do**
8: **if** $cosdis < cosdistance$ **then**
9: $cosdistance \Leftarrow cosdis$
10: **else**
11: *continue*
12: **end if**
13: **end for**
14: $s \Leftarrow s + 1$
15: **end while**
16: Calculate the center point between each document in the cluster after clustering and select the centroid as the new cluster center point F (center1, center2,..., centerK), 0<centerK<N+1;
17: **for** $j \Leftarrow 0$ to K **do**
18: $K \Leftarrow F$
19: **end for**
20: $flagcosdis \Leftarrow cos(K(centerK), F(centerK))$
21: **if** $flagcosdis = Min$ **then**
22: $Flag \Leftarrow FALSE$
23: **else**
24: *continue*
25: **end if**
26: **end while**
27: Return result

and F new cluster center points are selected(lines 15-17 in Algorithm 1).

4) Calculate the cosine distance of the new and old cluster centers, if the distance is close and the center point does not change, complete the clustering, otherwise return to the fifth line in algorithm 1 and repeat the above steps until the cluster center does not change, the end sign is Flag = FALSE (lines 18-23 in Algorithm 1).

C. MULTI-LEVEL INTELLIGENT CONTRACT AUTOMATIC GENERATION METHOD BASED ON DEEP LEARNING

Character-level Recurrent Neural Network (Char-RNN) [10], [22] is a model that operates on the character level, taking a large amount of character text as input, predicting the next character based on the previous one through RNN algorithm, and finally generating code text similar to the original text. However, this model suffers from problems such as gradient disappearance. To address this issue, this paper uses Bidirectional GRU (Bidirectional GRU) as the hidden layer to replace the basic RNN model. This model not only solves the problems that exist in the basic RNN model but also has a simpler structure and calculation than LSTM. It requires less computing power and improves efficiency.

The smart contract auto-generation model is shown in Figure 2. In this model, we train on word-level units, which is more effective in dimensionality reduction compared to character-level. The input layer uses word embedding instead of one-hot encoding, which enhances semantic accuracy and optimizes memory usage. Then, we use two layers of bidirectional GRU as the neural network layer, which can better utilize information from both forward and backward sequences. Finally, the output layer is a fully connected layer, where softmax learns the information from the bidirectional hidden layers of the GRU and outputs the probability of all word groups, achieving the output information of the output layer. Before generating the code, in order to improve the accuracy of the experiment, we conducted secondary preprocessing of the clustered smart contract and deleted the incomplete and incorrect codes in the code. In view of the nature of Solidity language features, identifiers, variables, and so on in the code are divided by space, so we divide words through space and establish a dictionary. We will encode the subsequent word input to the input layer in the way of word embedding. In this paper, the input of Embedding is a 2-dimensional tensor (sentence number, sequence length), the sequence length is set to 30, the output layer is 3000 words, and the forward GRU layer dimension and backward GRU layer dimension are 256 and 128 respectively.

In the above-mentioned automatic generation of deep learning code, the samples for training the model are all smart contract code samples, and the input is part of the samples. Through the input prediction of the model, the unified basic smart contract code is output. For example, the training sample is all the code samples similar to "function1" in the function layer cluster, the input is partly a code sample similar to "function1", and the output is a unified "function1" smart contract code. Similarly, we can generate multiple "function", "contract" and other hierarchical codes through this model.

$$\cos(i, center) = \frac{\sum_{i=1}^{n}(E_i \times E_{center})}{\sqrt{\sum_{i=1}^{n}(E_i)^2 \times \sum_{i=1}^{n}(E_{center})^2}} \times \lambda + \frac{\sum_{i=1}^{n}(N_i \times N_{center})}{\sqrt{\sum_{i=1}^{n}(N_i)^2 \times \sum_{i=1}^{n}(N_{center})^2}} \times (1-\lambda) \quad (3)$$

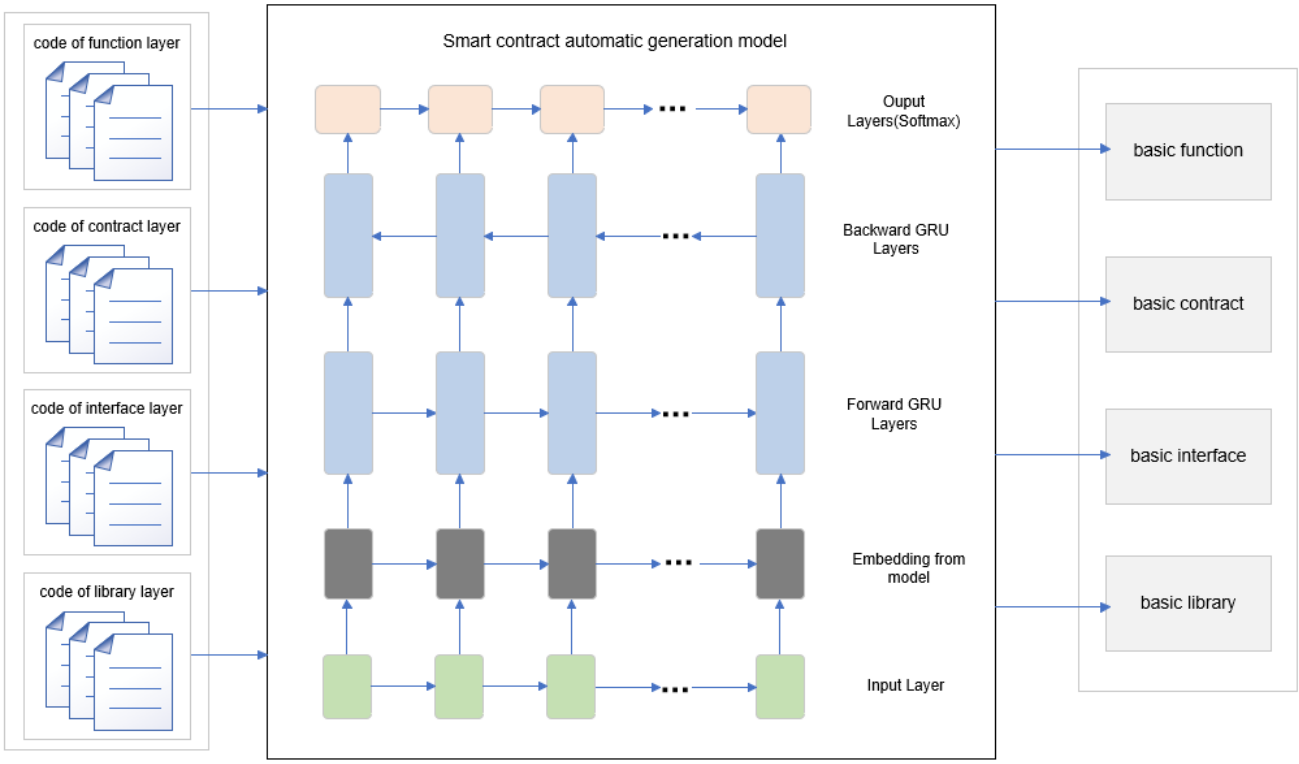

FIGURE 2. Smart contract automatic generation model.

D. ANNOTATION-GUIDED SMART CONTRACT MATCHING

Text similarity is to calculate the specific quantitative value between multiple texts through a certain metric standard, so as to measure their similarity. It is the link between basic research such as text modeling and representation and upper-level research on text potential information. The annotation covers a lot of code description information, based on which the similarity between the content of the annotations and the user's demand information is calculated, and finally, the smart contract code with the highest similarity is matched to the user. Jaccard [12] is a common way of similarity calculation, which is characterized by simple principles and suitable for short text calculation. In the process of generating smart contract code using a deep learning model, we stratified the smart contract code data set, so the corresponding number of individual annotation matching characters is relatively small, which can be considered as the type of short text, that is exactly in line with the computing characteristics of Jaccard. On the other hand, Jaccard is a commonly used similarity calculation method. Compared with other algorithms, the calculation amount and efficiency are better. In the adaptive matching process of smart contracts, the development time is better reduced and the requirements of the development environment are more friendly. So this paper adopts Jaccard as a measure, design-friendly interactive interface, developers through the interface input their requirements, through the calculation of various level generated code annotation information and the requirements of developers information similarity, call similar results optimal smart contract to developers, realize the function of adaptive matching smart contract code.

In formula 4, U is the code set,u_i is the ith code, and N is the number of codes. In formula 5, C is the demand information set of user.The similarity value is the ratio of the size of the intersection U of C and to the size of the union of U and C.

$$U = \{u_i, \quad i \in N\} \tag{4}$$

$$C = \{c\} \tag{5}$$

$$\text{Max}\left\{J\left(u_i, C\right) = \frac{|u_i \cap c|}{|u_i \cup c|} = \frac{|u_i \cap c|}{|u_i| + |c| - |u_i \cap c|},\ i \leq N\right\} \tag{6}$$

In formula 6, u_i represents any code in the database. We calculate the developer's demand information and code similarity, and finally find the largest similar code as the target code to interact with the developer. The specific logic is shown in Figure 3.

IV. EXPERIMENTS AND RESULTS ANALYSIS

A. EXPERIMENTAL ENVIRONMENT AND THE DATASETS

The experiments in this paper were conducted on a 64-bit Windows 10 operating system with an AMD Ryzen5-4500U with Radeon Graphics 2.38 GHz processor, 16.00GB of RAM, and Python 3.10 as the programming and running environment. The dataset used in this article was obtained through a self-write crawler program, selecting 10,437 smart

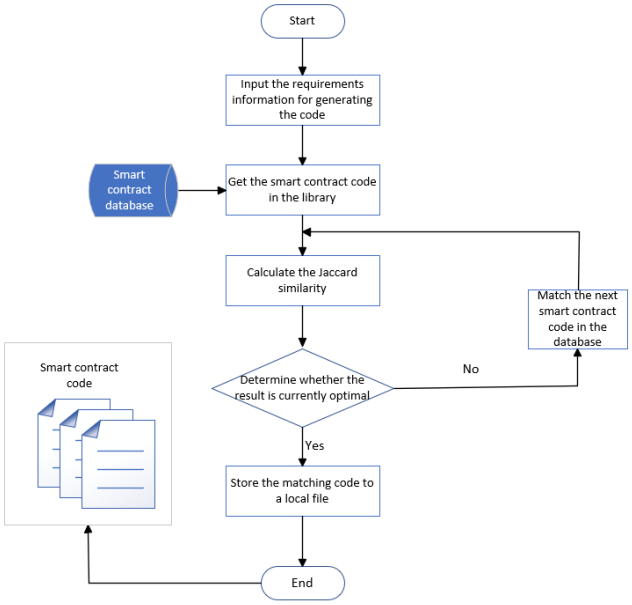

FIGURE 3. Smart contract adaptive matching.

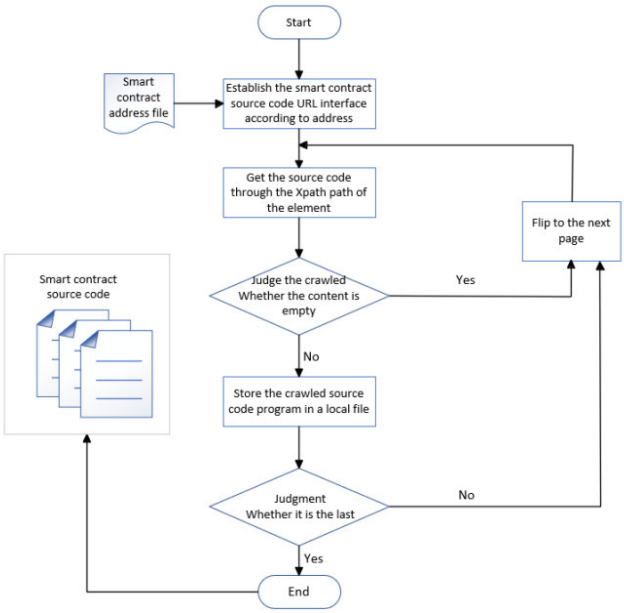

FIGURE 4. Smart contract crawl.

contract source codes from the Ethereum platform, with smart contract document sizes ranging from 1 to 211KB. We performed preliminary data preprocessing, removing some smart contracts in JSON and ABI formats and retaining 1,737 smart contract documents with sizes ranging from 50KB to 80KB as the dataset for this article's experiments. Finally, we divided the smart contract source code in the dataset into ''function,'' ''contract,'' ''interface,'' and ''library'' levels and divided the annotations and corresponding code bodies. We also defined the stop word document required for clustering analysis.

The primary step in this article is to obtain a dataset of clustered smart contracts. Ethereum is currently the most widely used blockchain interaction platform, and it has released numerous open-source licensed smart contract source codes. We obtained the latest smart contracts from the Ethereum platform as an experimental dataset by writing a smart contract source code crawler program. The flowchart of the smart contract acquisition process in this article is shown in Figure 4, and the specific operation process is as follows: download the smart contract address file from the Ethereum platform, open the Ethereum browser through an automated program, read the addresses in the file and construct the smart contract source code page URL, start from the first page, update the URL content in a loop, and obtain each page of smart contract source code and store it in a local file, until the smart contract address file is completely read.

As a widely used database, Mysql can not only help us map between smart contract code body and annotation content but also store long text type (long text) data. Therefore, establish the corresponding ''function'', ''contract'', ''library'', and ''interface'' table, each record in the table stores the code body and notes, establish a good mapping relationship, at the same time, through the code body and annotation field name, which greatly improves the convenience of code content management. Based on this we use the Python built-in Tkinter designed developers and smart contract code interactive interface, developers select the code level in the interface and input their demand information. Then, the Jaccard distance is calculated one by one based on the user demand information and the annotations corresponding to the smart contract codes at all levels stored in the database, and finally, the smart contract code with the best calculation result is selected and presented to the developer from the interface.

B. RESULTS AND ANALYSIS

In the clustering experiment, the index used by the elbow method is *SSE*, which is an index to measure the sum of squares of the distance between the data point in the cluster and the cluster center. The specific formula is shown as follows:

$$SSE = \sum_{i=1}^{k} \sum_{p \in M_i} |p - m_i| \tag{7}$$

where M_i is each cluster, p is the sample point in M_i, m_i is the central sample point in M_i, and SSE is the clustering error of all samples. The elbow method was carried out according to the above indicators, and k was finally obtained as 618, and the visualization results were shown in Figure 5:

After completing the clustering experiment, we obtained 618 clusters of smart contract codes as the dataset for the smart contract auto-generation model. Each code cluster contains many similar smart contract codes. Based on the model built in this paper, each code cluster generates a unique basic smart contract code with rich descriptive comments. As shown in Table 3, the generated hierarchical code includes

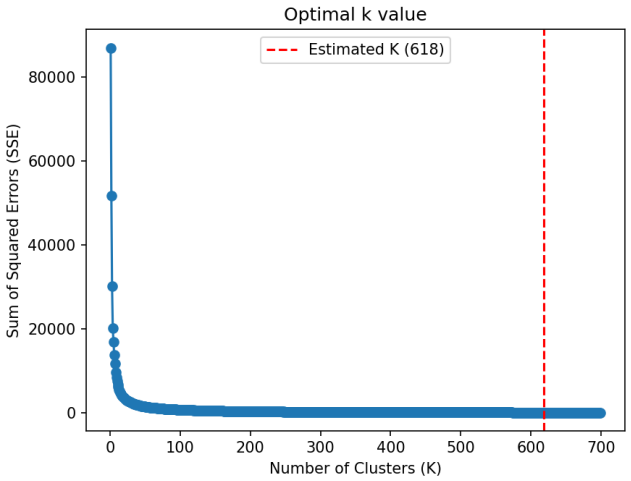

FIGURE 5. Optimal k value.

basic function code, basic contract code, basic interface code, and basic library contract code.

To ensure the validity of the generated code, we evaluated it using the BLEU metrics. The BLEU [18] indicator is a mechanism widely used in evaluation for machine translation tasks. Here we measure the quality of the generated code by the number (n-grams) between the generated code text and the reference text. In n elements, the value of n is 1 to 4, and the BLEU score ranges from 0 to 1. If the two sentences match perfectly, the BLEU is 1.0, otherwise, if the two sentences do not match perfectly, the BLEU is 0.0. The BLEU mechanism scoring formula is as formula 8:

$$BLEU = BP^{*} \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{8}$$

For the generated code text, in order to avoid the influence of symbols on the evaluation scores, we replaced symbols such as "+" and "−" with spaces. Furthermore, according to the characteristics of the Solidity language, we tokenized the word using spaces and obtained BLEU scores. We evaluated the generated code separately for each level using the BLEU mechanism. The experimental BLEU scores are shown in Table 2. The last row of the table shows the scores of existing experiments. From the data, it can be seen that the code generated by the above method has good quality. Compared with past experiments, average BLEU1-4 scores increased by 22.5%, 23.7%, 24.9%, and 37.2% respectively.

In addition, smart contract security checking is extremely important. In order to improve the security of the generated smart contracts, this paper uses formal verification and symbolic execution methods to detect the generated code, using the VaaS and Mythril tools respectively. In this experiment, unfinished generated code, duplicated code, and code that does not pass the detection are all considered incorrect smart contract codes. Table 4 shows the accuracy of each tool after detection, and the accuracy of existing experiments is 68.27%. Compared with this experiment, the

TABLE 2. The BLEU experimental score for generating the smart contract code.

Code layer	BLEU-1	BLEU-2	BLEU-3	BLEU-4
function	0.794	0.753	0.722	0.690
contract	0.672	0.613	0.565	0.526
interface	0.867	0.850	0.838	0.825
library	0.767	0.730	0.699	0.666
exist experiment	0.632	0.595	0.530	0.493

TABLE 3. Smart contracts automatically generate results.

Code layer	Generated base code	Number
function	transferFrom· · · approve	473
contract	Context· · · ERC20	67
interface	IERC20· · · ISupply	53
library	Address· · · SafeMath	25

TABLE 4. Generate the smart contract code detection results.

Code layer	VaaS check	Myhril check
function	73.3%	62.4%
contract	72.3%	71.9%
interface	75.6%	71.9%
library	82.3%	76.4%
exist experiment	68.27%	

detection tool in this paper is more comprehensive and the average accuracy rate is increased by 11.5%.

This paper designs a friendly interactive interface, as shown in Figure 6. Developers input their requirements information in this interface and realize the generation of adaptive code matching by matching the smart contract with the greatest similarity from the database and delivering it to the developers. Here we take the example by generating a "myToken" case, where "myToken" is a token contract based on the ERC20 standard.

In order to verify the feasibility of the proposed scheme, we compare the experiment with the latest technology in the field of natural language processing. The excellent text-processing capability of the large model has attracted the attention of many scholars around the world. Chat-GPT, as the most outstanding general class large model at present, plays a very strong prospect in the aspects of text classification, human-machine dialogue, etc. The model is much better than other models in all aspects and has a very good reference value. Second, CodelLama is currently a relatively excellent open-source code generation large model in addition to GPT, and its effect is also at the top of the list of foreign large models and opens its weight for researchers to learn from. In addition, CodeGeex and Starcoder, as a new generation of large code generation models, also show a relatively good performance in code completion, generation, and other capabilities. As the hottest and most advanced technology at present, large models can better help this paper to evaluate and compare the experimental effect, and this paper conducts experimental comparison with the above four large models. This paper designs a friendly interactive

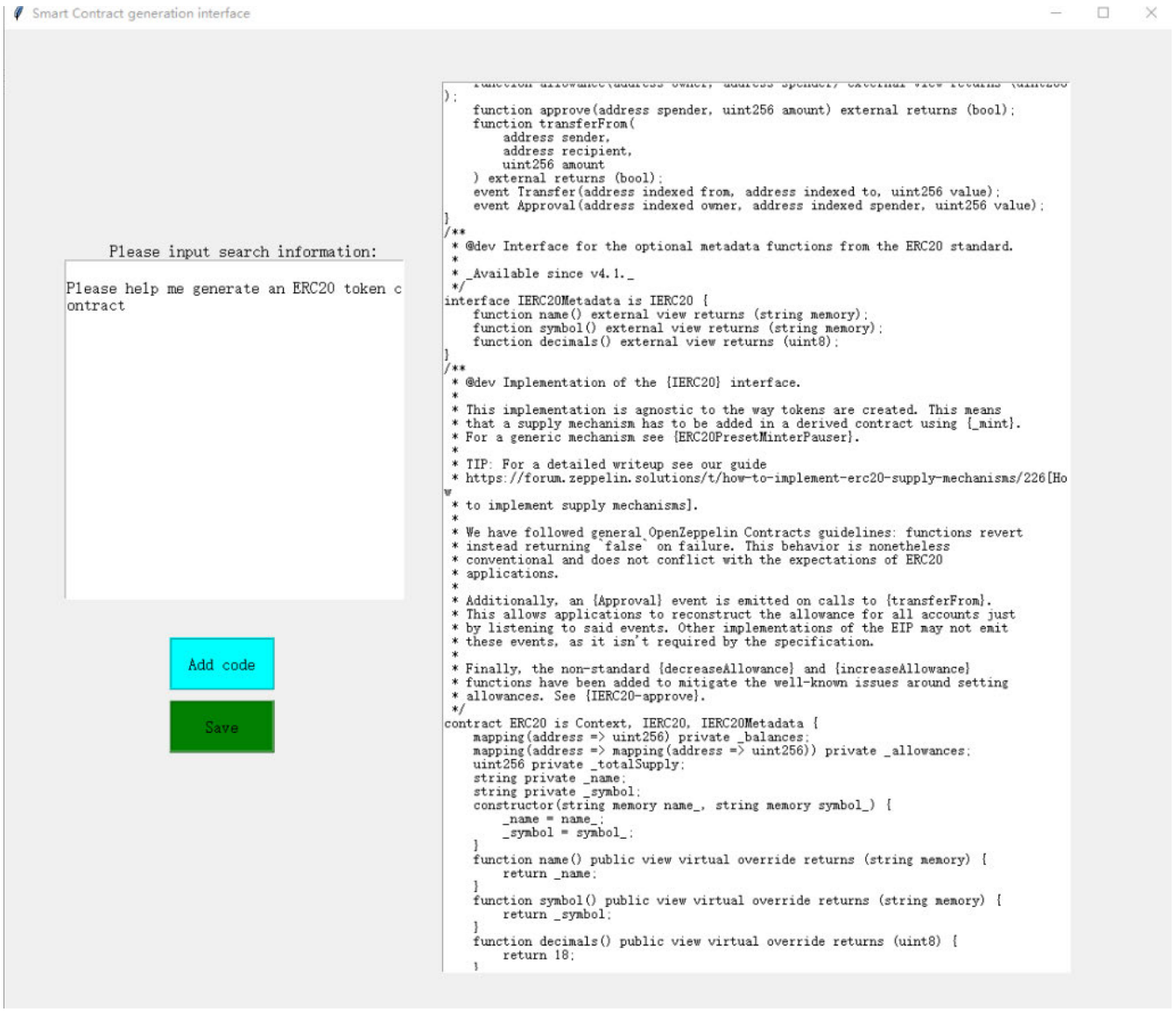

FIGURE 6. Automatically generate the "ERC20" token contract.

interface, as shown in Figure 6. Developers input their own requirements information in the interface and realize the generation of adaptive code matching by matching the most similar smart contract from the database and delivering it to the developer.

In order to ensure the fairness of the experiment, the same instructions were input into the system designed in this paper, and Chatt-GPT, CodelLama, CodeGeex, and Starcoder, and tested 30 times respectively (that is, 30 instructions were used to let the model help generate the corresponding smart contract code). Here we take the example of generating ''myToken'', where ''myToken'' is a token contract based on the ERC20 standard. As shown in Figure 6, the user enters ''Please help me generate an ERC 20 token contract'' in the ''Please enter search information'' input box of the smart contract generation interface. After clicking the ''Add code'' button, the smart contract code generated in the text box on the right side of the interface is the object code that adaptively matches the user's demand information. As can be seen from the code generated in the figure, this paper not only realizes the generation of ERC20 code and other required contract code but also interacts with the corresponding annotation content with the interface, thus improving the user's programming friendliness. Finally, by clicking the ''Save'' button in the interface, we name the file ''myToken'' and store it in the computer using the. sol file format. Here, the ERC 20 token contract is generated in ChatGPT with the same password, as shown in Figure 7.

After testing the 30 instructions, in order to verify the validity of the generated smart contract, the source code is used as the standard code in BLEU, and the smart contract code generated in this paper is compared and tested with the

```scss
pragma solidity ^0.8.0;

contract MyToken {
    string public name;
    string public symbol;
    uint256 public totalSupply;
    mapping(address => uint256) public balanceOf;
    mapping(address => mapping(address => uint256)) public allowance;

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 v

    constructor(string memory _name, string memory _symbol, uint256 _totalSu
        name = _name;
        symbol = _symbol;
        totalSupply = _totalSupply;
        balanceOf[msg.sender] = _totalSupply;
        emit Transfer(address(0), msg.sender, _totalSupply);
    }

    function transfer(address _to, uint256 _value) public returns (bool succ
        require(_to != address(0), "Cannot transfer to zero address");
        require(_value <= balanceOf[msg.sender], "Insufficient balance");
        balanceOf[msg.sender] -= _value;
        balanceOf[_to] += _value;
        emit Transfer(msg.sender, _to, _value);
        return true;
    }

    function approve(address _spender, uint256 _value) public returns (bool
        require(_spender != address(0), "Cannot approve zero address");
        allowance[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);
        return true;
    }

    function transferFrom(address _from, address _to, uint256 _value) public
        require(_to != address(0), "Cannot transfer to zero address");
        require(_value <= balanceOf[_from], "Insufficient balance");
        require(_value <= allowance[_from][msg.sender], "Insufficient allowa
        balanceOf[_from] -= _value;
        balanceOf[_to] += _value;
        allowance[_from][msg.sender] -= _value;
        emit Transfer(_from, _to, _value);
        return true;
    }
}
```

FIGURE 7. ChatGPT generate the "ERC20" token contract.

code generated by Chate-GPT, CodelLama, CodeGeex, and Starcoder respectively. First, we generate code in each model with the same password. Next, we calculated and compared the average BLEU score between the generated code and the source code for each model, as shown in Table 5. Table 5 shows the experimental schematics of this paper compared with Chat-GPT, CodelLama, CodeGeex, and Starcoder. From the table's score, the average score of BLEU 1-4 is very close to that of ChatGPT and CodelLama. The generated code is closer to the source code, and the effect is better. Compared with CodeGeex and Starcoder, the score of the scheme in this paper is significantly improved. This shows that the large model that ranks top in the current large model list has a very good performance in code generation, and this scheme has better capability than the general large model of code generation. Secondly, the code generated in this paper

TABLE 5. The experimental results of this scheme are compared with those of large model.

BLEU	BLEU-1	BLEU-2	BLEU-3	BLEU-4
This scheme	0.674	0.643	0.620	0.601
ChatGPT 3.5	0.712	0.703	0.689	0.611
CodelLama	0.701	0.692	0.663	0.632
CodeGeex	0.589	0.537	0.501	0.493
Starcoder	0.521	0.499	0.431	0.401

contains the relevant interface code and library contract code. The code generated by the large model is currently limited to a single contract layer code, and the structure is relatively simple, which can not meet the various needs of current smart contract development. On the other hand, the code generated in this article provides a lot of annotation information, which is one of the focuses of this article. Comment information can help developers quickly and efficiently understand code and improve development efficiency.

C. COMPUTATIONAL COMPLEXITY

The computational complexity involved in this paper includes three parts, the first part is the time complexity and space complexity of the Kmeans algorithm; The second part is the complexity of GRU model training (loss change) and the time complexity and space complexity of the GRU model. The third part is the time complexity and space complexity used in the similarity calculation algorithm. Time complexity is a function that qualitatively describes the running time of the algorithm, which is defined as the number of repeated executions of basic statements in the algorithm is a function $f(n)$ of the problem size n, and the time measurement of the algorithm is denoted as:

$$T(n) = O(f(n)) \tag{9}$$

It indicates that with the increase of problem size n, the growth rate of algorithm execution time is the same as that of $f(n)$, which is called the asymptotic time complexity of the algorithm, referred to as time complexity. As for the storage space requirement of the algorithm, similar to the time Complexity of the algorithm, we use the asymptotic Space Complexity as a measure of the storage space required by the algorithm, referred to as the space complexity, which is also a function of the problem size n, denoted as:

$$S(n) = O(f(n)) \tag{10}$$

According to the analysis of the Kmeans algorithm in this paper, the maximum number of layers of cyclic iteration in the algorithm is 3 layers. If the problem scale of each layer is n, then the maximum time complexity of the algorithm is $T(n) = O\left(n^3\right)$. The space used is 2 sets of sample points. Suppose the problem size of each set is n, then the maximum space complexity of the algorithm is $S(n) = O(n) + O(n) \approx O(n)$. According to the above analysis, the computational complexity in the clustering algorithm includes time complexity and space complexity, which are cubic order $O\left(n^3\right)$ and linear order $O\left(n^2\right)$respectively.

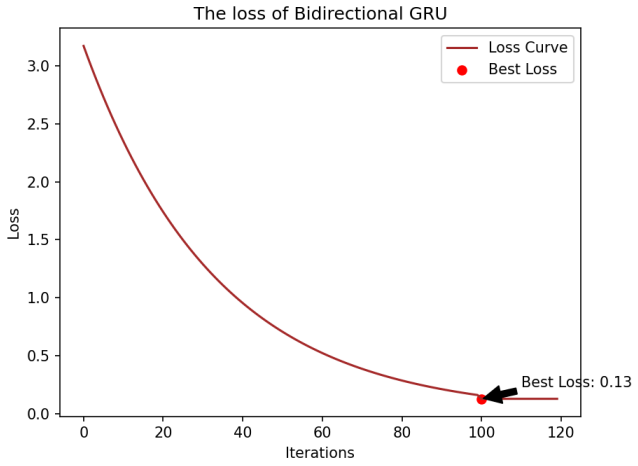

FIGURE 8. Model training complexity.

In the code generation model, this paper adopts bidirectional GRU as the training model, and "loss" is an indicator used to measure the difference between the model prediction and the actual target. The smaller the loss, the closer the predicted result of the model is to the actual target, while the larger the loss, the greater the difference between the predicted result and the actual target. Therefore, loss shows the complexity of GRU model training in this paper. The model loss transformation of this scheme is shown in Figure 8:

According to the change of loss in the figure, when the loss value approaches 0.13, the loss will basically no longer decline. Therefore, we believe that when the loss=0.13, the model will be trained to the best, otherwise there will be problems of overfitting and underfitting. In addition, the bidirectional GRU is used as the model for code generation, and the complexity of the model includes time complexity and space complexity.When calculating the time complexity, the main concern is the computation cost per moment. Assuming that the sequence length is T, the hidden layer size is N, the input dimension is M, and the model receives input characters size is n, then the time complexity of each GRU unit is $O(N \cdot M + N^2)$. Since the model is bidirectional, the maximum time complexity of the model is $T(n) = 2O\left(n \cdot N \cdot M + n \cdot N^2\right) \approx O\left(n^3\right)$. When calculating the space complexity, we mainly focus on the number of parameters and the storage of the intermediate state. Suppose that the number of parameters of each GRU unit is $O(N \cdot M + N^2)$, and the space complexity of the intermediate state is $O(T \cdot N)$. Since the model is bidirectional, the maximum space complexity of the model is $S(n) = 2O\left(n \cdot N \cdot M + n \cdot N^2\right) + 2O(T \cdot N \cdot n) = O\left(n^3\right)$.To sum up, the complexity of model training in this paper is loss=0.13, and the time complexity and space complexity of the model is cubic order $O\left(n^3\right)$ and $O\left(n^3\right)$ respectively.

The final computational complexity is the time complexity and space complexity of the similarity calculation algorithm. In Part III-D of the paper, the process of similarity calculation is explained in detail. When each user's demand instruction is issued, the smart contract code in the library will be matched successively until the code with the greatest similarity is obtained. Assuming that the size of the code in the library is n, then each time the code is generated, n similarity calculations will be performed, so the maximum time complexity is $O(n)$. The problem size of the samples in one of the sample sets is considered as n, then the maximum space complexity is $O(n)$. To sum up, in the process of similarity calculation, the computational complexity includes time complexity and space complexity, which are divided into specific linear order $O(n)$ and linear order $O(n)$.

V. THE DEFICIENCY OF THIS ARTICLE

This paper proposes that the automatic generation scheme of smart contracts based on code annotations shows better efficiency and User-friendliness in writing smart contracts. Although it has a good application prospect and value in the development of smart contracts in finance, education, and other fields, there are still some limitations. Firstly, annotations are added to both the clustering algorithm and similarity calculation in the scheme proposed in this paper, and the quality of annotations will directly affect the effect of this scheme. When the obtained annotation content is of low quality or wrong, the accuracy of the clustering algorithm and similarity calculation will be greatly reduced, thus resulting in inaccuracies and errors in matching smart contracts to users. Secondly, this topic focuses on the research of general-purpose smart contract codes. The model we designed can better generate smart contract code commonly used in most fields, but when faced with more detailed problems in a certain field, the generated code may not be able to meet the needs of users. Therefore, the requirements for smart contract code in different fields will be more subtle differences, and the system designed in this paper can not meet the specific code requirements in a single domain. To sum up, in the face of the above situation, the effectiveness of this study is reduced to a certain extent, and it is only applicable to the condition that annotations ensure high quality and the development needs of general-purpose smart contract code. The scheme proposed in this paper can effectively provide users in various fields with safe, reliable, and commonly used smart contract code.

VI. CONCLUSION AND FUTURE ENHANCEMENT

This article proposes an intelligent contract automatic generation solution based on code annotations. The solution obtains smart contract source code data by crawling Ethereum and divides it into hierarchical levels and code annotation segments. In the clustering analysis, similar code is effectively extracted from each hierarchical level from the perspectives of comments and code bodies. Meanwhile, a deep learning model based on character-level recurrent neural networks (Char-RNN) is used, which replaces the hidden layer with

bidirectional gated recurrent units (bi-directional GRU) and constructs a smart contract automatic generation model to generate unified code for each hierarchical level. Regarding smart contract security, the BLEU scoring mechanism and smart contract detection tools are used to evaluate the quality and security of the generated code, and a convenient and fast visualization interface is designed for developers to use. The experimental results show that the quality and security indicators of the smart contract code generated by this solution are good, and compared with previous experiments, there is a significant improvement.

The scheme in this paper can be effectively applied to the application scenario of smart contract development, which has certain research significance for the efficiency and security of smart contract development. However, there are certain deficiencies in the work of this paper. First of all, this paper does not conduct research on smart contracts in specific fields. At present, it is limited to contract codes commonly used in various fields and lacks research on codes in specific fields. Secondly, although the experimental results of the smart contract automatic generation model used in this paper have been significantly improved, the construction of the model can still be optimized in follow-up research, and there is still much room for improvement in its accuracy and security. Further exploration and research based on the above issues will be considered in the future.

REFERENCES

[1] A. Meri and M. Alzahrani, "Utilization of block chain technology and smart contracts in the education procedure of universities," *Int. J. Intell. Syst. Appl. Eng.*, vol. 12, no. 12s, pp. 652–661, 2024.

[2] S. Bag, S. K. Kumar, and M. K. Tiwari, "An efficient recommendation generation using relevant Jaccard similarity," *Inf. Sci.*, vol. 483, pp. 53–64, May 2019.

[3] A. Bas, M. O. Topal, Ç. Duman, and I. van Heerden, "A brief history of deep learning-based text generation," in *Proc. Int. Conf. Comput. Appl. (ICCA)*, Dec. 2022, pp. 1–4.

[4] A. Bhardwaj, S. B. H. Shah, A. Shankar, M. Alazab, M. Kumar, and T. R. Gadekallu, "Penetration testing framework for smart contract blockchain," *Peer Peer Netw. Appl.*, vol. 14, no. 5, pp. 2635–2650, Sep. 2021.

[5] C. Zhanqi, M. G. W. Zan, C. Xiang, and Y. Guang, "Summary of automated generation methods for code annotation," *J. Softw.*, vol. 32, no. 7, p. 24, 2021.

[6] L. Corallo, G. Li, K. Reagan, A. Saxena, and A. S. Varde, and B. Wilde, "A framework for german-english machine translation with GRU RNN," in *Proc. EDBT/ICDT Workshops*, 2022, pp. 1–8.

[7] Y. Dong, X. Jiang, Z. Jin, and G. Li, "Self-collaboration code generation via ChatGPT," Tech. Rep., 2023.

[8] Z. Zhao, G. Yichen, and Z. Bin, "Research and implementation of the automatic smart contract generation method for Ethereum," *J. East China Normal Univ. Natural Sci. Ed.*, vol. 2020, no. 5, p. 21, 2020.

[9] M. Geng, S. Wang, D. Dong, S. Gu, F. Peng, W. Ruan, and X. Liao, "Fine-grained code-comment semantic interaction analysis," in *Proc. IEEE/ACM 30th Int. Conf. Program Comprehension (ICPC)*, May 2022, pp. 585–596.

[10] L. Bixin, B. Yihao, H. Tianyuan, and L. Zecheng, "Summary of contract security and privacy security research on smart contract," *J. Comput. Sci.*, vol. 12, p. 44, Jan. 2021.

[11] T. Iqbal and S. Qureshi, "The survey: Text generation models in deep learning," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, no. 6, pp. 2515–2528, Jun. 2022.

[12] F. J. Ming, "Jaccard application of the improved algorithm in the detection of user entity behavior analysis," *Comput. Appl. Softw.*, vol. 39, no. 2, p. 5, 2022.

[13] S. Xiaohong J. Yuan, and M. Chenguang, "Safety defect reporting identification by combining noise filtering and deep learning," *J. Comput. Sci.*, vol. 8, p. 45, Jun. 2022.

[14] G. Khodabandelou, H. Moon, Y. Amirat, and S. Mohammed, "A fuzzy convolutional attention-based GRU network for human activity recognition," *Eng. Appl. Artif. Intell.*, vol. 118, Feb. 2023, Art. no. 105702.

[15] Y. Li, Q. Wang, T. Xiao, T. Liu, and J. Zhu, "Neural machine translation with joint representation," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, 2020, pp. 8285–8292.

[16] D. Mao, F. Wang, Y. Wang, and Z. Hao, "Visual and user-defined smart contract designing system based on automatic coding," *IEEE Access*, vol. 7, pp. 73131–73143, 2019.

[17] A. Mirzal, "Statistical analysis of microarray data clustering using NMF, spectral clustering, kmeans, and GMM," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 19, no. 2, pp. 1173–1192, Mar. 2022.

[18] M. M. Misgar, F. Mushtaq, S. S. Khurana, and M. Kumar, "Recognition of offline handwritten Urdu characters using RNN and LSTM models," *Multimedia Tools Appl.*, vol. 82, no. 2, pp. 2053–2076, Jan. 2023.

[19] L. Yong, W. Y. P. Bin, and L. Zheng, "Automated generation method of code annotation based on the convolutional neural network," *Comput. Sci.*, vol. 48, no. 12, p. 8, 2021.

[20] H. Qinming, H. Butian, T. Zhengzheng, W. Xun, Q. Peng, and L. Zhenguang, "Research review on smart contract security vulnerability detection technology," *J. Softw.*, vol. 8, p. 33, Jan. 2022.

[21] P. Rani, M. Birrer, S. Panichella, M. Ghafari, and O. Nierstrasz, "What do developers discuss about code comments?" in *Proc. IEEE 21st Int. Work. Conf. Source Code Anal. Manipulation (SCAM)*, Sep. 2021, pp. 153–164.

[22] C. Sendner, H. Chen, H. Fereidooni, L. Petzi, J. König, J. Stang, A. Dmitrienko, A.-R. Sadeghi, and F. Koushanfar, "Smarter contracts: Detecting vulnerabilities in smart contracts with deep transfer learning," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2023, pp. 1–18.

[23] R. Sharma, F. Chen, and F. Fard, "LAMNER: Code comment generation using character language model and named entity recognition," in *Proc. IEEE/ACM 30th Int. Conf. Program Comprehension (ICPC)*, New York, NY, USA, May 2022, pp. 48–59.

[24] H. Shi and D. Zhao, "License plate recognition system based on improved YOLOv5 and GRU," *IEEE Access*, vol. 11, pp. 10429–10439, 2023.

[25] S. Siddiqui, S. Hameed, S. A. Shah, A. K. Khan, and A. Aneiba, "Smart contract-based security architecture for collaborative services in municipal smart cities," *J. Syst. Archit.*, vol. 135, Feb. 2023, Art. no. 102802.

[26] X. Su, Y. Hu, W. Liu, Z. Jiang, C. Qiu, J. Xiong, and J. Sun, "A blockchain-based smart contract model for secured energy trading management in smart microgrids," *Secur. Privacy*, vol. 7, no. 1, p. e341, 2024.

[27] P. Tolmach, Y. Li, S.-W. Lin, Y. Liu, and Z. Li, "A survey of smart contract formal specification and verification," *ACM Comput. Surv.*, vol. 54, no. 7, pp. 1–38, Sep. 2022.

[28] F. Ullah and F. Al-Turjman, "A conceptual framework for blockchain smart contract adoption to manage real estate deals in smart cities," *Neural Comput. Appl.*, vol. 35, no. 7, pp. 5033–5054, Mar. 2023.

[29] Á. J. Varela-Vaca and A. M. R. Quintero, "Smart contract languages: A multivocal mapping study," *ACM Comput. Surv.*, vol. 54, no. 1, pp. 1–38, Jan. 2022.

[30] Z. Wan, X. Xia, D. Lo, J. Chen, X. Luo, and X. Yang, "Smart contract security: A Practitioners' perspective," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, May 2021, pp. 1410–1422.

[31] W. Wang, H. Huang, Z. Yin, T. R. Gadekallu, M. Alazab, and C. Su, "Smart contract token-based privacy-preserving access control system for industrial Internet of Things," *Digit. Commun. Netw.*, vol. 9, no. 2, pp. 337–346, Apr. 2023.

[32] F. Zhenyuan, W. Qian, and W. Cheng, "Review of k-means clustering algorithm studies," *Electron. Des. Eng.*, vol. 20, no. 7, p. 4, 2012.

[33] M. Xia, H. Shao, X. Ma, and C. W. de Silva, "A stacked GRU-RNN-based approach for predicting renewable energy and electricity load for smart grid operation," *IEEE Trans. Ind. Informat.*, vol. 17, no. 10, pp. 7050–7059, Oct. 2021.

[34] Q. Zhu, Z. Sun, Y.-A. Xiao, W. Zhang, K. Yuan, Y. Xiong, and L. Zhang, "A syntax-guided edit decoder for neural program repair," 2021, *arXiv:2106.08253*.

[35] W. Zou, D. Lo, P. S. Kochhar, X. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu, "Smart contract development: Challenges and opportunities," *IEEE Trans. Softw. Eng.*, vol. 47, no. 10, pp. 2084–2106, Oct. 2021.

YONG CHEN was born in Lengshuijiang, Hunan, in 1986. He received the Ph.D. degree in computer software and theory from Wuhan University, in 2013.

He was a Senior Engineer. Notably, he was a Research Assistant with the City University of Hong Kong, from June to December 2012. His academic and research expertise encompass several key domains, including big data auditing, blockchain, software reliability, compilation optimization, and embedded systems. He has offered courses in ''Big Data Processing Techniques,'' ''Fundamentals of Computer Systems,'' ''Big Data Auditing,'' and ''C Language Program Design.'' Among his noteworthy achievements, he has taken the helm in leading various research projects, including Jiangsu Provincial High Education Natural Science Research Project, from 2020 to 2022. He has authored a significant academic work titled ''Research on Green Compilation Theory and Methods for Embedded Systems,'' which he serves as the second author, published by the Science Press in May 2014. His research contributions are evident through his numerous publications in respected journals, including *International Journal of Electronics* and *Audit Research*. Notably, his work in *International Journal of Electronics* has been acknowledged with an article on software defect prediction. Furthermore, he holds two granted invention patents, one for a method of automatically generating audit questions based on compliance issues and another for a high-reliability knowledge base construction method for querying and tracing extensive unstructured data content. Additionally, he boasts two software copyrights, one for data dependency analysis software based on LLVM IR and another for an Audit Big Data Thematic Knowledge Base System. His academic and research achievements underscore his commitment to advancing the fields of computer science and audit technology.

DEFENG HU was born in March 1998. He is currently pursuing the master's degree with the School of Computer Science, Nanjing Audit University.

He has made notable contributions to the field of computer science with publications in prominent journals. Furthermore, he is currently actively engaged as an Intern in the field of natural language processing with Yunwen Network Technology Company, Nanjing. His primary research interests include natural language processing, with a specific focus on code generation and the domain of big data auditing. He is committed to advancing knowledge in these areas and contributing to the academic community.

CHAO XU received the Ph.D. degree.

He is currently a Distinguished Scholar. He is a Professor who also serves as a Ph.D. Supervisor. He has earned recognition as a Teaching Master in Jiangsu. He has extended his expertise to the international arena as a Visiting Scholar with Columbia University, USA. He is a Trusted Consultant in the realm of big data with the Ministry of Public Security. His influence also extends to academic leadership, where he holds a position as the Director of the National Association of Computer Education in Higher Education and Jiangsu Provincial Computer Society. Additionally, he is a Dedicated Supervisor with Jiangsu Provincial Cryptography Society. At present, he holds the prestigious position of the Dean of the School of Computer Science/Intelligent Audit, Nanjing Audit University. His remarkable contributions to the field are further exemplified by his role as an Expert with the Big Data Working Group, International Organization of Supreme Audit Institutions (INTOSAI). He also assumes the role of the Executive Dean of the Institute of Audit Science and Technology. Furthermore, he concurrently serves as the Dean of the Greater Bay Area Audit Research Institute. Furthermore, they have been sought after as a consultant in the field of audit information technology, providing expertise, and guidance to various governmental bodies, including the Ministry of Public Security, the National Audit Office, the State Administration of Taxation, the National Immigration Administration, and government audit agencies at all levels.

NANNAN CHEN was born in December 2000. She is currently pursuing the master's degree with the School of Computer Science, Nanjing Audit University. She has demonstrated a strong commitment to academic excellence, with her research primarily focused on Human–Computer Interaction and Interpersonal Trust. Her dedication to these fields is underscored by her publications in prominent computer science journals, contributing to the ongoing advancement of knowledge in these areas.