IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 5, 2019, accepted May 28, 2019, date of publication June 4, 2019, date of current version June 18, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2920776

Visual and User-Defined Smart Contract Designing System Based on Automatic Coding

DIANHUI MAO[1,2], FAN WANG[1], YALEI WANG[1], AND ZHIHAO HAO[1,3]

[1]Beijing Key Laboratory of Big Data Technology for Food Safety, College of Computer and Information Engineering, Beijing Technology and Business University, Beijing 100048, China
[2]National Engineering Laboratory for Agri-Product Quality Traceability, Beijing Technology and Business University, Beijing 100048, China
[3]Pattern Analysis and Machine Intelligence Group, Department of Computer and Information Science, University of Macau, Macau 999078, China

Corresponding author: Fan Wang (wfan0601@163.com)

This work was supported in part by the National Social Science Fund of China under Grant 18BGL202, in part by the National Natural Science Foundation of China Grant 61877002, in part by the Beijing Natural Science Foundation under Grant 4172013, in part by the Social Science and Humanity on Young Fund of the Ministry of Education under Grant 17YJCZH127, and in part by Beijing Municipal Commission of Education under Grant PXM2019 014213 000007.

ABSTRACT Smart contract applications based on Ethereum blockchain have been widely used in many fields. They are developed by professional developers using specialized programming languages like solidity. It requires high requirements on knowledge of the specialized field and the proficiency in contract programming. Thus, it is hard for normal users to design a usable smart contract based on their own demands. Most current studies about smart contracts focus on the security of coding while lack of friendly tools for users to design the specialized templates of contracts coding. This paper provides a visual and user-defined smart contract designing systems. It makes the development of domain-specific smart contracts simpler and visualization for contract users. The system implements the domain-specific features extraction about the crawled data sets of smart contract programs by TF-IDF and K-means++ clustering algorithm. Then, it achieves the automatic generation of unified basic function codes by Char-RNN (improved by LSTM) based on the domain-specific features. The system adopts Google Blockly and links the generated codes with UI controls. Finally, it provides a set of specialized templates of basic functions for users to design smart contracts by the friendly interface. It reduces the difficulty and costs of contract programming. The paper offers a case study to design contracts by users. The designed contracts were validated on the existing system to implement the food trading and traders' credit evaluation. The experimental results show that the designed smart contracts achieve good integration with the existing system and they can be deployed and compiled successfully.

INDEX TERMS Smart contract, Char-RNN, LSTM, automatic coding.

I. INTRODUCTION

Decentralized cryptocurrencies such as Bitcoin [1] have rapidly gained popularity in recent years. Blockchain [2] is regarded as the underlying technique support and the fundament of Bitcoin system. Smart contract [3] is just like a piece of software or computer program in the blockchain network and it is similar to a contract in the physical world. Smart contracts are pre-written logic of the computer codes by Turing-complete languages such as Solidity, Golang and so on [4]. Then users can execute these programs ''if this happens then do that'', run and verified by participants to ensure trustworthiness. As we all know, blockchains give users distributed trustworthy storage. Smart contracts are stored and replicated on the blockchains that inherit the blockchains' properties: Smart contracts remove reliance on a third party when establishing business relations so that the parties making an agreement can transact directly with each other.

With the development of blockchain, smart contract applications [5] have been developed and researched in recent years. And they are mainly used in two aspects: electronic monetary transactions and other general applications for storing information. For example, Bogner et al. [6] developed

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

a smart contract application aiming to rent devices. The food supply field applied the smart contracts to achieve food trade [7], [8]. It decreases the costs of traders on the food supply chain and provides a decentralized, untrustworthy, accountable and transparent architecture based on blockchain technology. Other than applications listed above, Al-Bassam et al. [9] built a system for identity management.

Ethereum [10] as one example of the decentralized platforms that provides a container called Ethereum Virtual Machine (EVM) to execute smart contracts. Being a permissionless ledger [11], anybody is free to join the Ethereum network, perform transactions and access the contract's source code. However, smart contracts which are executed on the blockchain just like a tool of hackers for illegal benefits. The numerous incidents on Ethereum blockchain [12] in recent times prove this: The DAO (Distributed Autonomous Organization), King of Ether, Rock Paper Scissors game, Governmental, FirePonzi etc. Previous incidents show that the security, privacy and normalization of smart contracts must be focused during the process of contract codifying. Aiming at these existing issues of smart contracts, Alharby and Moorsel [13] introduced many researches on optimization of smart contract codes. In these researches, for avoiding code errors and security vulnerabilities of smart contracts, some tools such as ''SmartCheck'' [14], ''Oyente'' [10], ''Zeppelin'' [15] and so on were developed to detect or tackle codifying and security issues of smart contracts.

Through the analysis of these existing researches of smart contracts, this paper offers other two main challenges about the existing programming problems of smart contracts:

A. THE DESIGNABILITY OF CODING

As mentioned above, Solidity is a strict language for contract coding. It has stringent coding requirements for contract programming. And a lot of knowledge and proficiency in a specialized field of smart contract are necessary for contract designing. Contract designers have to figure out the code logic in advance for designing or running the smart contract of their own application. It not only overloads the developers' works and leads to inefficiencies of coding, but also is unfriendly for junior-level developers or non-developers such as normal contract users to implement a usable smart contract designing based on their own demands.

B. THE SPECIALIZED TEMPLATES OF BASIC FUNCTIONS CODING BASED ON DOMAIN-SPECIFIC FEATURES

Smart contracts applications have been widely used in many fields such as financial, medical, payment transactions and so on. With the rapid development of smart contract applications, the number of contract codes on Ethereum platform increases dramatically. However, the business logic and requirements of smart contracts are significantly different from each field. Therefore, the specialized templates of basic functions based on domain-specific features are beneficial to decrease the complexity of contract designing. While, there is a lack of the domain-specific approach for contract designers to implement the targeted contracts coding.

Concerning above two challenges, this paper provides an innovative system named ''Visual and User-defined Smart Contract Designing System Based on Automatic Coding''. It provides users an easier, cost-effective and interactive approach to design smart contracts (written by Solidity language). The system not only reduces costs of developers but also offers a designability way of contract coding for junior-level developers or normal users. Firstly, the system crawls the data sets of smart contract programs from an Ethereum blockchain explorer. After that the system obtains the domain-specific features about the crawled data sets by TF-IDF and clustering algorithm. Then the system focused on a specialized domain (such as the domain of trading) to learn the features of contracts by Char-RNN [16]. Besides, system adopts LSTM as the cell of recurrent neural network to replace the basic RNN of Char-RNN for better performance. And the unified and standardized basic function codes of contracts are generated by improved Char-RNN model. Eventually, the system offers a visual and interactive editor based on Google Blockly for users to design the customized smart contracts. Users design their contract programs based on their own definition and requests by draggable UI controls rather than writing code. The UI controls which show on the editor page wire up the generated basic function codes. After the process of designing is complete, the system provides the functionality as ''Save'' for user-defined smart contract.

The contributions of this paper can be summarized in the following four points:

1) The system implements a method to extract smart contracts' domain-specific features. Because of the widespread of smart contracts, the system implements the features extraction about the crawled collection of smart contracts. The system adopts TF-IDF and K-means++ clustering algorithm to extract the features. And the crawled collection of smart contracts is divided into different categories based on domain-specific features.
2) The system implements an approach to automatically generate the basic functions of specialized templates for smart contract coding. Compared with the basic RNN, LSTM supports time steps and it provides a way to address the time-series prediction problem. LSTM attaches great importance to content-based features rather than local text information. Thus, the system adopts LSTM [17] cell of recurrent neural network to replace the basic RNN of Char-RNN model. The system focuses on the domain-specific of smart contracts and implements the automatic coding by improved Char-RNN. With this method, the unified and standardized basic function codes of smart contracts based on the domain-specific features are generated.
3) The system offers a visual web interface for users to implement the user-defined contract coding.

The system adopts Google Blockly and provides a designability interface for users' interactions. The generated codes by improved Char-RNN are embedded into the UI controls on the visualized interface. And users can design smart contracts based on their own personalized requirements by dragging and dropping these UI controls. This way of smart contract designing reduces the difficulty in programming and save the time for users.

4) This paper offers a case study to verify the effectiveness of the designed contracts by the system for food trading. This paper adopts the framework [18] named ''Credit Evaluation System Based on Blockchain'' which implemented the food trading and traders' credit evaluation on the blockchain network. And the result presents that generated contract programs achieve a good integration with the existing Credit Evaluation System.

This paper is organized as follows: in section 2, the paper focuses on the discussion of the system by introducing the related work about contribution of other researchers on this topic. In section 3, the workflow of the system is illustrated. It introduces the components of the system in details. Section 4 provides an introduction of the implementation of the system. And the results of the system are showed in section 5. Section 6 presents the conclusions and future directions of efforts about the system.

II. RELATED WORK

The purpose of this section is to provide a brief overview of existing studies of smart contracts and mainly focus on the studies of the contract programming problems during the smart contract application development process.

Bitcoin [1] as a peer-to-peer electronic cash system was proposed in 2009 by Nakamoto and quickly captured the public's attention and interest. With the development of this cryptocurrency, blockchain [2], [3] regarded as the underlying technique support and the fundament of Bitcoin rapidly becomes one of the hot research topics in computer application area. A huge number of emerging altcoins such as Ripple [19] and Litecoin [20] accelerate the development of blockchian. Additionally, several blockchain platforms such as Ethereum and Hyperledger have been extended to provide supports for blockchain applications by ''smart contracts'' [4].

Smart contracts [5] are user-defined computer programs in fact. A smart contract, also known as a cryptocontract, is an automatable and enforceable agreement that directly controls the transfer of digital currencies or assets between parties under certain conditions. A smart contract not only defines the rules and penalties related to an agreement in the same way that a traditional contract does, but it can also automatically enforce those obligations. Automatable by computer, although some parts may require human input and control. It is enforceable either by legal enforcement of rights and obligations or via tamper-proof execution of computer code.

Ethereum [10], [11] is the most popular development platform of blockchain. It provides the built-in fully-fledged Turing-complete programming language such as Solidity language that can be used to create ''smart contracts'' to encode arbitrary state transition functions and create systems for blockchain applications. A smart contract is executable code that runs on the EVM [21], [22] to facilitate, execute and enforce the terms of an agreement between untrusted parties. It can trigger data reads and writes, do expensive computations like using cryptographic primitives, make calls (send messages) to other contracts, etc. For example, two insurance companies, Atlas Insurance in Malta and Axa in France, tested smart contracts in 2017. They had prototypes that compensated airline customers if their flights were delayed. Health systems use smart contracts to record and safely transfer data. They can transfer patient data in a secure way, allowing no access from third parties. For governments, smart contracts running on the blockchain can make voting systems completely trustless and much more secure. Smart contracts have broad range of other applications [6], [7], such as identity systems, decentralized file storage systems, etc. In addition to storing users' identities or file data, smart contracts also can handle and transfer assets of considerable value. They also can be applied in the token systems or financial applications [8], [9]. They are usually used to pay for transaction fees [23], [24]. All transactions that have ever occurred in the Ethereum can be recorded in the blockchain network. And compared to traditional applications, smart contract applications do not rely on a trusted third party to operate, resulting in low transaction costs.

However, smart contracts are ''immutable''. Once they are deployed, their code is impossible to change, making it impossible to fix any discovered bugs. According to a study of smart contract issues by Alharby and Moorsel [13], the challenges of smart contracts mainly focus on the four aspects: codifying, security, privacy and performance issues, a series of attacks which exploit the security vulnerabilities of Ethereum smart contracts have happened in the past time. For example, on 17th June 2016, hackers exploited the programming loophole in the DAO and siphoned away one-third of the DAO's funds. That's around 50 million dollars. For avoiding attacks like DAO in the feature, it is more important to ensure the security, privacy and normalization of contract codifying. A lot of researches have been studied on optimization of contract codes [12] and many tools were developed. For instance, ''SmartCheck'' [14] is a static code analyzer developed by SmartDec Security Team. It runs analysis in Solidity source code and automatically checks smart contracts for security vulnerabilities and bad practices. Similarly, ''Slither'' also provides a static analysis framework with detectors for many common Solidity issues. ''Oyente'' is a tool to analyze smart contract code and find common vulnerabilities. And some auditing tools [15] such as

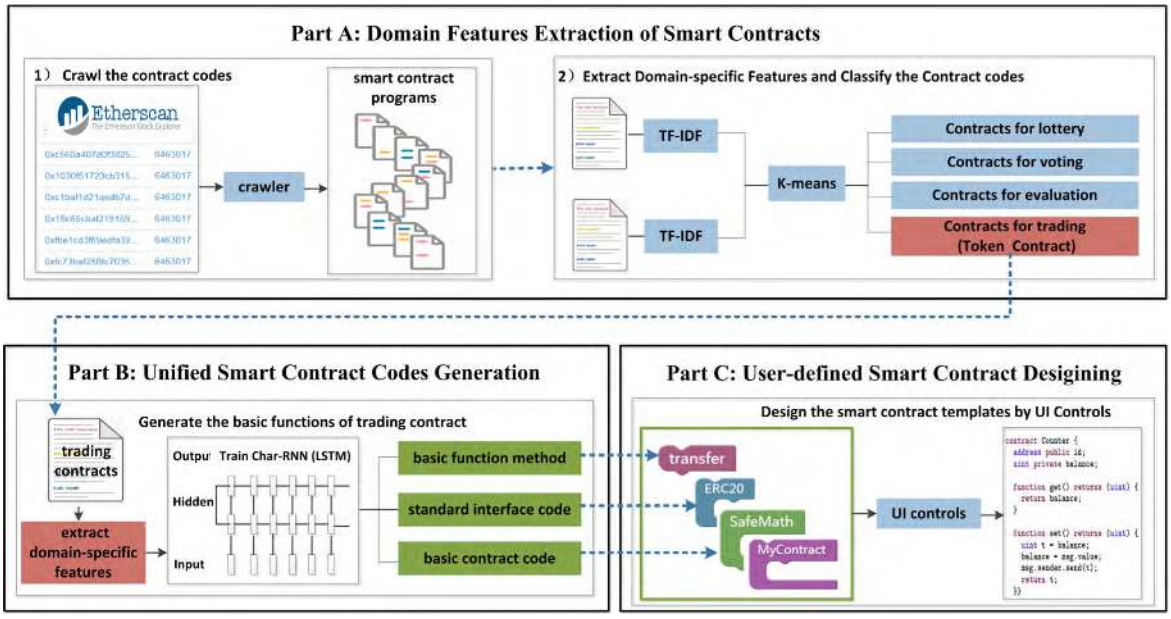

FIGURE 1. The architecture of the visual and user-defined smart contract designing system.

"Solidified.io", "Zeppelin", "Token Market" and so on aims to analyze the code for intended behavior, security or Solidity construct usage. These researches identify bugs in the system and checks whether the contracts operate as intended. Besides, a group of researchers from the University of Stanford, together with Visa Research, have created a mechanism to enhance the privacy in smart contracts from the second largest crypto blockchain in the market, Ethereum. A paper describing the mechanism was published on Stanford University's Applied Cryptography Group website in February, 2019. However, these researches basically aimed at security and privacy issues to normalize the codifying process of smart contracts. As the most fundamental issue of smart contract coding, there is a little focus on programming-friendly. There are still many challenges in the development of smart contract programming such as the designability of contract coding and the specialized templates of contract coding.

Different from previous works, this paper provides a visual and user-defined smart contract designing system for users. The system proposed in this paper decreases the costs for entry-level developers on good programming of smart contracts. And the system also offers a friendly and interactive codes editor by graphical interface for non-developer to design personalized contracts. The system embeds the automatic generated basic function codes into UI controls that decreasing the complexity of coding for users. They don't need to understand the code logic and they just design smart contracts by dragging UI controls based on their own business definitions and requests.

III. WORKFLOW OF THE SYSTEM

For users who are inexperience in smart contract programing, the system of "Visual and User-defined Smart Contract Designing System Based on Automatic Coding" makes it possible for them to implement the smart contract designing. The system provides an approach to generate unified and standardized basic function codes of the contracts based on the domain-specific features. The system also provides a set of template controls by visualized and interactive way for users to design the smart contract programs and link the generated basic function codes with the UI controls. The system serves users' needs and offers a visualized system with strong advantages that reduces the coding costs and complexity.

This section introduces the basic architecture of the smart contract designing system. And the following figure shows the architecture of the system in details.

As shown in Fig. 1, the system includes three main parts: Part A implements the domain features extraction of smart contracts by TF-IDF and K-means++ clustering algorithm; Part B implements the unified smart contract codes generation based on a specific domain feature such as the field of trading. In this part, the basic function codes are generated by improved Char-RNN model; Part C implements the visual and user-defined smart contract designing by UI controls. The details of each part are as follows:

A. PART A: DOMAIN FEATURES EXTRACTION OF SMART CONTRACTS

Smart contracts have been widely used in many fields. The system proposed in this paper aims at a specific field of contract applications and implements the extraction of domain-specific features of smart contract codes firstly. Then the system classifies the crawled collection of smart contracts based on domain-specific feature. Two steps of this part are performed as follows.

1) **Crawl the contract codes:** This step firstly fetches the verified smart contract programs (written by Solidity language) as data collection from the Ethereum Blockchain Explorer named Etherscan [25] by crawler program. It provides the data set for extracting the domain-specific features of smart contracts in next step.
2) **Extract Domain-specific Features and Classify the contract codes:** In above step, the collection of smart contracts in various fields is crawled. Aiming at widespread contracts, this step divided the crawled collection of contract programs into different categories based on their domain-specific features. First, the feature vectors of smart contracts are extracted and weighted by TF-IDF method. Then, cluster the extracted features and classify them into different categories by the clustering algorithm K-means++. For example, the categories concern the contracts for trading (such as Token contract), or the contracts for voting, lottery et al.

B. PART B: UNIFIED SMART CONTRACT CODES GENERATION

From previous part, the system acquires the classification results based on the domain-specific features of smart contract collection. Among of the classified results, the system chooses the contracts for trading field and implements the unified contract codes generation in this paper. The system offers a set of specialized templates of basic functions for users to design the smart contracts more efficiently and easily. First, the trading contract programs obtained in Part A are regarded as the dataset. Then the dataset is trained by Char-RNN algorithm. Meanwhile, the system adopts LSTM as recurrent neural network to replace the basic RNN of Char-RNN model. LSTM is beneficial to learn the domain-specific features and provide a better performance for smart contract codes generation. The improved Char-RNN model obtains the domain-specific features of trading contracts by training dataset. Finally, improved Char-RNN model generates the unified and standardized basic function codes of trading contracts eventually. The generated codes of basic functions include the basic function method (e.g., ''transfer'' function), standard interface code (e.g., ''ERC20'' code) and basic contract code (e.g., ''SafeMath'' contract code).

C. PART C: VISUAL AND USER-DEFINED SMART CONTRACT CODES DESIGNING

When all steps in above two parts are completed, to facilitate accessibility for users, the system provides a visual page editor to help users write smart contracts.. There are a set of UI controls on the page. Users can drag and drop these controls to design the business logic of smart contracts based on their requirements quickly and easily. In addition, these UI controls wire up the generated basic function codes of the trading contract. For example, if a user wants to design a smart contract named ''MyContract'' for food trading, the user can define his or her demands by dragging the UI controls on the page. If the user chooses the control of ''transfer'', the code of ''transfer'' function appears in the code-editor box on the page. The system provides a good designability for users to program the smart contracts. And it reduces the difficulty for contract designing.

IV. IMPLEMENTATION OF THE SYSTEM

A. DOMAIN FEATURES EXTRACTION OF SMART CONTRACTS

Ethereum [10], [11] is the most popular platform for developing the smart contract applications. As a distributed public blockchain network, Ethereum is an open source platform that making the source codes of smart contract applications available. This open source environment greatly facilitates the communication of developers and it promotes the development of contract applications. Thus, the system adopts Ethereum as the operation environment. And the system fetches the smart contract programs as data sets from the Ethereum blockchain explorer named Etherscan firstly. Then the system implements the domain features extraction and classification of crawled data sets based on the domain-specific features (or Part A which is shown in Fig. 1). The implementation process consists of two steps: crawl the contract codes and extract domain-specific features (classify the contract codes) respectively which is introduce in in details in the following subsection.

1) CRAWL THE CONTRACT CODES

The huge numbers of data collection about smart contract programs is necessary in this study of the system. While, if the number of data set is not enough it will limit the representation of the contract features. However, a publicly smart contract program data set is not yet available. Thus, to analysis the domain-specific features about different fields of smart contracts better, the first step of the system crawls the smart contract programs as data collection from Etherscan.

Etherscan [25] is a block explorer and analytics platform for Ethereum, a decentralized smart contracts platform. It hosts a collection of web-based tools for exploring the public Ethereum network, based on the transactions that have been confirmed on the Ethereum blockchain. As a blockchain explorer, Etherscan can only provide and display information on transactions that occur on the Ethereum blockchain. It can't process transactions and troubleshoot transaction failures. The entire flow for collecting the smart contract codes by crawler program from Etherscan is illustrated in Fig. 2. It starts from the first index page of smart contracts on the web and the URL lists of contracts are founded. Then the current index page jumps to the details page of smart contract program. All programs of contracts are saved into the folders until the last contract on the URL lists. And the same operation is executed repeatedly on the next page. Finally, the data collection of smart contract programs is acquired.

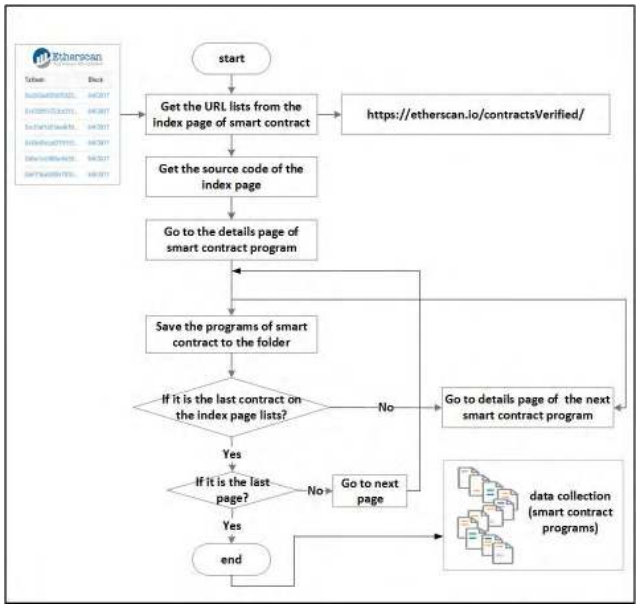

FIGURE 2. The flow for crawling the smart contract codes.

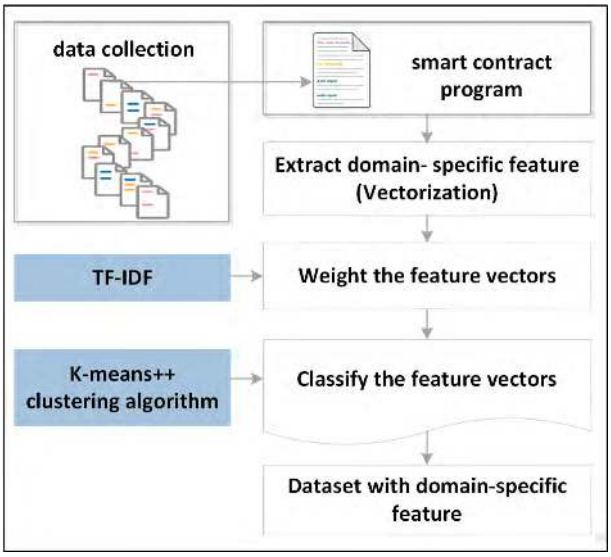

FIGURE 3. The flow for extracting domain-specific features and classfying the contracts.

2) EXTRACT DOMAIN-SPECIFIC FEATURES AND CLASSIFY THE CONTRACTS

Smart contract applications are commonly used in diversified fields. Different applications have different implementations of smart contracts. Thus, features extraction is crucial for better understanding the smart contract programs and designing a contract with specialized domain feature. For crawled smart contracts collection, this step aims to extract and classify the smart contract programs based on their domain-specific features. The method of contracts features extraction and classification adopts TF-IDF and K-means++ algorithm in the system.

TF-IDF [26] is often used as a weighting factor in searches of information retrieval, text mining, and user modeling [27], [28]. It is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The TF-IDF value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. TF-IDF is one of the most popular term-weighting schemes that can be successfully used for stop-words filtering in various subject fields, including text summarization and classification. Typically, it is composed by following two terms.

a: TF: TERM FREQUENCY, WHICH MEASURES HOW FREQUENTLY A TERM OCCURS IN A DOCUMENT

Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length as a way of normalization.

$$TF(trading) = S_t/Sum \tag{1}$$

In Equation (1), S_t represents the number of times term ''trading'' appears in a smart contract program, Sum is the total number of terms in the contract.

b: IDF: INVERSE DOCUMENT FREQUENCY, WHICH MEASURES HOW IMPORTANT A TERM IS

While computing TF, all terms are considered equally important. However, it is known that certain terms, such as ''public'', ''return'', and other words in the programs, may appear a lot of times but have little importance. Thus, the frequent terms need to be weighed down while scale up the rare ones, by computing the following:

$$IDF(trading) = lg(Sum/D_t) \tag{2}$$

In Equation (2), D_t represents the number of smart contracts with the specific term ''trading'' in it.

The crawled contract program can be regarded as the text document. It contains the keyword for describing the contract. For example, with above method, it can be examined how frequent certain words such as ''trading'' appear in a contract and can determine what may be important to this contract. As shown in Fig. 3, the ''feature vectors'' about the domain-specific of the trading contract are formed. And the vectorization of above contract data collection is implemented by training word vectors through Google's open source word2vec. Then, the feature vectors are weighted according to the frequency and importance by TF-IDF model. TF-IDF model gives a high weight to words that are used frequently by trading contracts, but not used as frequently overall.

K-means++ is a simple and fast clustering algorithm to cluster and classify the unsupervised data. It plays an important role in the research and analysis of a large number of unlabeled data such as the crawled smart contract programs in

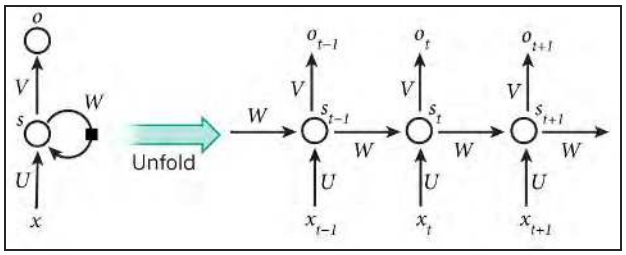

FIGURE 4. The workflow of the standard RNN.

the system. The implementation process of the K-means++ clustering algorithm is as follows:

1) Select the appropriate value of K by K-means++ algorithm. It determines categories of the smart contract programs based on the domain-specific features of contracts.
2) Calculate the distance of each data with the cluster centroid K and Locate the closest centroid of each contract program.
3) Reculate and determine the centroid of each cluster.
4) Repeat steps 2) and 3) until the cluster centroid doesn't change again.

Ultimately, the system integrates the value of weighting which is calculated by TF-IDF model into K-means++ clustering algorithm. Based on domain-specific features, the system classifies the smart contracts into K categories.

B. UNIFIED SMART CONTRACT CODES GENERATION

Smart contract program consists of different function codes to for user's operation. For example, trading contract contain the function methods such as "transfer", standard interface such as "ERC20" for token, basic contract such as "SafeMath" for computation during the trading process. In order to make the process of smart contract designing become easier and more normalized for users. The system adopts improved Char-RNN algorithm to generate the unified basic functions of contracts based on the domain-specific features.

Recurrent Neural Networks (RNNs) [29], [30] work best on sequence tasks. As shown in Fig. 4, RNN has a high-dimensional hidden state with nonlinear dynamics that enable them to remember and process past information powerfully.

The standard RNN is formalized as follows: Given a sequence of input vectors $(x_1, \cdots, x_T)$, the RNN computes a sequence of hidden states $(h_1, \cdots, h_T)$ and a sequence of outputs $(o_1, \cdots, o_T)$by iterating the following equations:

$$h_t = tanh\left(Ux_t + Wh_{t-1} + b_h\right), \quad t \in [1, T] \tag{3}$$

$$o_t = Vh_t + b_o \tag{4}$$

In above equations, U, W, V is the weight matrix of each gate shown in Fig. 4. U is the input-to-hidden weight matrix, W is the hidden-to-hidden (or recurrent) weight matrix, V is the hidden-to-output weight matrix, and the vectors b_h and b_o are the biases of each gate. The parameters U, W, V and b need to be trained.

Char-RNN is character-level language models. It uses recurrent neural network aiming at the automatic generation of temporal text [31]. It takes a huge chunk of text file as an input and feeds it into the basic RNN algorithm that learns to predict the next character in the sequence. After training the basic RNN, it can generate text character by character that looks stylistically similar to the original dataset. While, the basic RNN face the increasingly difficult problems such as vanishing gradient and exploding gradient. These problems are bad for Char-RNN algorithm to retain information of text file through the time steps and generate the text character.

Instead of the basic recurrent neural network in Char-RNN model, the system applies the Long Short Term Memory (LSTM) network [17] as the cell of RNN for better performance to generate the basic functions of smart contract. Compared with general RNN, LSTM can solve the issues of vanishing gradient and exploding gradient about basic RNN. In the hidden layer of a simple LSTM, it adds a cell state except the hidden states. And LSTM network also adds three gates (input gate, forget gate, output gate) to store information for a certain period of time better. The mathematical theories of the LSTM cell in hidden layer are as follows:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{5}$$

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{6}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{7}$$

In above equations, i_t, f_t, and o_t respectively represent the input, forget and output feature matrix. σ is the sigmoid function, W and b respectively represent the weight matrix and bias

When the input word x_t is given, the current cell state c_t and hidden state h_t can be updated as follows:

$$g_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{8}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{9}$$

$$h_t = o_t \odot tanh(c_t) \tag{10}$$

In above equations, $\odot$ represents the multiplication of matrix elements, g_t is the current information of the cell.

The improved Char-RNN model of the system adopts two hidden layers of LSTM network to replace basic RNN. The hidden layer of each LSTM consists of cell state c_t and hidden state h_t. And the size of the state is 128. Besides, the system set the maximum input size of LSTM is 3500 and the learning rate is 0.05. As the flow shown in Fig. 5, among of the classified smart contracts, the system uses the trading contract dataset as input for improved Char-RNN model. And it returns the generated basic functions (such as "transfer", "ERC20" "SafeMath" and so on) as output by improved Char-RNN model.

C. USER-DEFINED SMART CONTRACT CODES DESIGNING

In the Part C, the system offers a visualized interface for users to design smart contracts by UI controls. The interactive interface is based on Google Blockly [32] which is

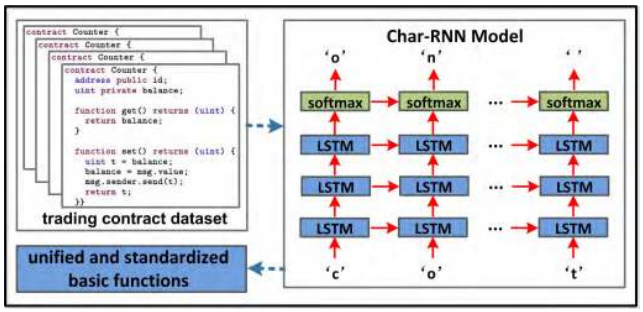

FIGURE 5. The flow for Char-RNN (LSTM) model to generate the unified and standardlized basic functions.

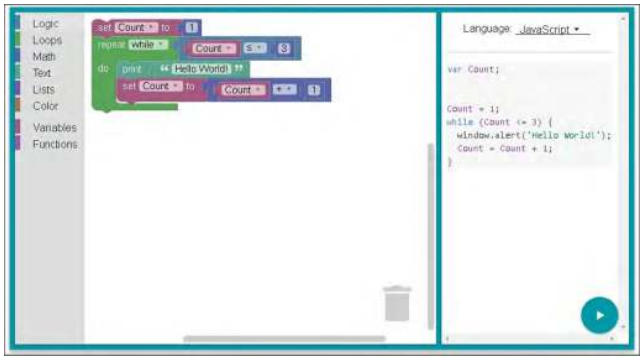

FIGURE 6. The demo of google blockly.

a web-based, visual programming editor shown in Fig. 6. It reduces the difficulty and costs for users to program the contracts.

Blocks shown in Fig. 6 are composed of three components:

1) *Block definition object:* Defines the look and behavior of a function block, including the text, color, fields, and connections.
2) *Generator function:* Generates the code string for the function blocks. It is always written in JavaScript.
3) *Blockly editor:* Toolbox of the editor is used to show the function blocks for users. Workspace of the editor is used to implement the code generation for users by adding the blocks into the workspace.

Based on the Google Blockly, many researchers focused on the particular programming language such as C, Python, PHP and so on to develop the programming editor by defining the block object and generator function. The system in this paper adopts the Ethereum blockchain as platform. Solidity is the most popular programming language for developing the Ethereum applications. Different from previous researches, the system implements smart contract designing focused on the language of Solidity based on Blockly. The system also offers the programming-friendly web interface for users.

In the system, Blockly is used to load blocks for web interface in the form of UI controls. And on this basis, the block object and generator function for smart contract

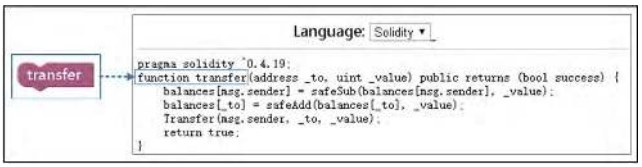

FIGURE 7. The block of the "transfer" function.

TABLE 1. Experimental environment.

Environment	Details
PC	Intel (R) Xeon (R) CPU 2.40 GHz (2 Processors), 12.0 GByte Memory
OS	Ubuntu Linux 16.04 LTS
Language	Nodejs-10.15.0, pyquery-1.4.1, Python 3.5, Solidity-0.4.19
Containerization	ethereumjs/testrpc-6.0.3, solc, truffle

codes are defined in the Part C via script files. The system creates a set of new JavaScript files for basic function codes about smart contracts. The basic function codes are generated by improved Char-RNN as above mentioned for trading contracts. These codes mainly contain three types such as the basic function method (e.g., "transferFrom" function), standard interface code (e.g., "ERC20" code) and basic contract code (e.g., "SafeMath" contract code). For example, Fig. 7 shows the snapshot for creating the block of "transfer" function. First, the new JavaScript file of "transfer" function needs to be included in the list of "script" tags in the editor's HTML file. And the look of the "transfer" function is defined in the JavaScript file. Once defined, the block can be referenced to the toolbox by the type name. Then, to transform the block into code, the system needs to pair the block with a generator function. Finally, users can design the contract codes through the web interface by dragging the UI controls of the "transfer" block. When the block is dragged from the toolbox to the workspace, the listening event of the generator is triggered. At the same time, the generator is associated to the generated codes of "transfer" function. And the function codes of "transfer" appear on the web interface for users.

V. RESULTS

In this section, the experimental environment of the system is introduced as follows. All components and functions of the system are developed for Ethereum applications. The experiment about crawler program for collecting smart contracts is performed by Python and PyQuery. We designed an experiment that used a crawler program written by Python and PyQuery to collect smart contracts. And PyQuery is a jquery-like library for python. The experiment about improved Char-RNN model for generating the contract codes is performed by Python 3.5. And it adopts Keras package with Tensorflow backend as the structure. More details of the basic environment are given in Table 1.

TABLE 2. Experimental dataset.

Categories of smart contracts	Quantity
Evaluation contracts	946
Lottery contracts	2254
Trading contracts	2532
Voting contracts	1808
The number of total contracts	7540

A. RESULTS OF DOMAIN FEATURES EXTRACTION OF SMART CONTRACTS

By performing the crawler program, the data collection of 30837 smart contract programs is crawled from Etherscan. The crawled contracts are written by Solidity language and they had been verified by Etherscan. And the size of them is between 1-745KB. The experiment preprocesses the dataset and selects 7562 contracts whose size is between 3-5KB as experimental data collection. Data preprocessing removes the space, comments and stop words of the contract programs in the dataset. The stop words are defined by Python program in the experiment such as ''string'', ''function'', ''public'' and so on.

Except for the dirty data, the domain features of 7540 contracts are extracted and classified by TF-IDF and K-means++. The size of the experimental data collection is 29.4MB. And the data collection of smart contracts is divided into following categories such as voting contracts, lottery contracts, evaluation contracts and trading contracts finally. The number of smart contracts with different domain-specific features is given in Table 2. The largest number of them is the dataset of trading contracts.

B. RESULTS OF UNIFIED SMART CONTRACT CODES GENERATION

Among of the categorized contract datasets in Table 2, the system chooses trading contract dataset as the study subject. It is a collection of 2532 smart contracts for trading on the blockchain network. The basic structure of dataset's source program contains function methods such as ''transfer'', standard interface such as ''ERC20'' for token, basic contract such as ''SafeMath'' for computation during the trading process. Then the experiment adopts improved Char-RNN algorithm to learn and train the features of trading contract codes. Ultimately, it utilizes the trained deep learning model to generate the basic functions of trading contracts. The results of codes generation by improved Char-RNN model mainly contain three types of the basic functions written by Solidity language, as follows.

1) BASIC FUNCTION METHOD

The generation results of ''basic function method'' consist of five functions for performing basic operations during the process of trading.

1) The function of ''transfer'' allows the sender (i.e. the person who called this function) to transfer the number of his tokens to the account address of another person.
2) The function of ''transferFrom'' allows a trading contract transfers the number of tokens from a person's account address to another person's account address. This function is usually called by the contracts, not by people.
3) The function of ''approve'' allows the spender to spend approved number of tokens on your behalf.
4) The function of ''balanceOf'' returns the balance of the specified address.
5) The function of ''allowance'' tells how many tokens the owner has allowed the spender to spend.

```
function transferFrom(address _from, address _to, uint _value) public returns (bool success) {
    var _allowance = allowed[_from][msg.sender];
    balances[_to] = safeAdd(balances[_to], _value);
    balances[_from] = safeSub(balances[_from], _value);
    allowed[_from][msg.sender] = safeSub(_allowance, _value);
    Transfer(_from, _to, _value);
    return true;
}
```

FIGURE 8. The generation results of ''transferFrom'' function.

```
contract ERC20{
  uint public totalSupply;
  function balanceOf(address who) public constant returns (uint);
  function allowance(address owner, address spender) public constant returns (uint);
  function transfer(address toAddress, uint value) public returns (bool ok);
  function transferFrom(address fromAddress, address toAddress, uint value) public returns (bool ok);
  function approve(address spender, uint value) public returns (bool ok);
  event Transfer(address indexed fromAddress, address indexed toAddress, uint value);
  event Approval(address indexed owner, address indexed spender, uint value);
}
```

FIGURE 9. The generation results of ''ERC20'' interface.

Fig. 8 shows an example about the generation result (''transferFrom'' function) of ''basic function method''.

2) STANDARD INTERFACE CODE

ERC20 is a standard for tokens. It is not the only token standard, but it is the most popular one. It defines a common interface for smart contracts storing users' balance of fungible tokens, sometimes, referred to as a cryptocurrency. Fig. 9 shows the generation result (''ERC20'' interface) of ''standard interface code''.

3) BASIC CONTRACT CODE

SafeMath is a solidity math library especially designed to support safe math operations (for addition or subtraction). Safe means that it prevents overflow when working with ''uint''. Fig. 10 shows the generation result (''SafeMath'' contract) of ''basic contract code''.

C. RESULTS OF SMART CONTRACT DESIGNING

In the experiment, the generated results by improved Char-RNN mainly contain three types of the unified and standardlized basic function codes for trading contract. For example, they contain the basic function method (e.g., ''transferFrom'' function), standard interface code (e.g., ''ERC20'' code) and basic contract code (e.g., ''SafeMath'' contract code). To simplify the operation of users, these codes are embedded into the UI controls by Google Blockly and the system is integrated as a visualized page editor. Users can

```
contract SafeMath {
function safeSub(uint a,uint b)pure internal returns (uint){
  assert(b<=a);
  return a-b;
}
function safeAdd(uint a,uint b)pure internal returns (uint){
  uintc=a+b; assert(c>=a && c>=b);
  return c;
}
function safeMul(uint a,uint b)pure internal returns (uint){
  uintc=a*b;
  assert(a==0 || c/a==b);
  return c;
}
function safeDiv(uint a,uint b)pure internal returns (uint){
  assert(b>0);
  uintc=a/b;
  assert(a==b*c+a%b);
  return c;
 }
}
```

FIGURE 10. The generation results of "SafeMath" contract.

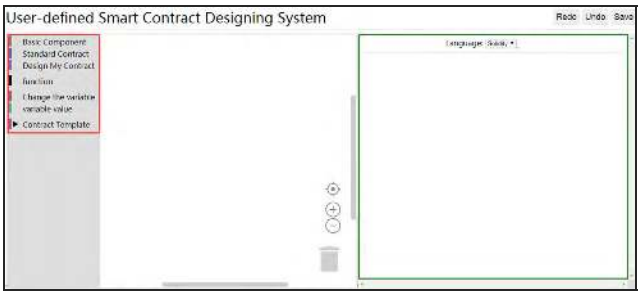

FIGURE 11. The interface of the integrated system.

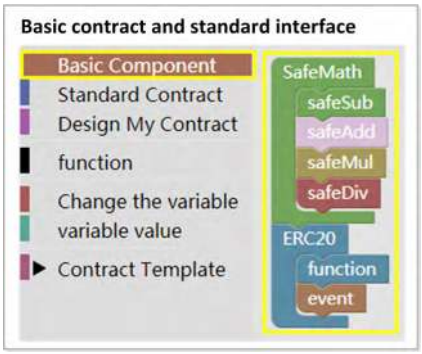

FIGURE 12. The UI controls for designing the basic contract and standard interface.

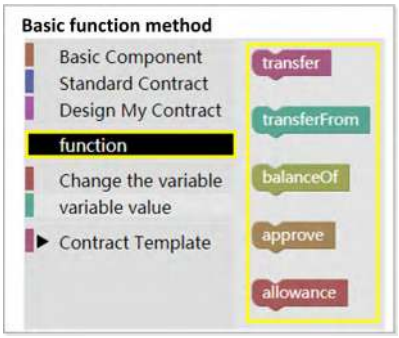

FIGURE 13. The UI controls for designing the basic function method.

drag the code blocks and design their contracts by this editor. This section introduces the interface of the entire designing system and describes the operating process for designing the smart contracts.

1) THE INTERFACE OF THE INTEGRATED SYSTEM

The system implements the user-defined smart contract designing. And Fig. 11 shows this visualized page editor of the system.

Users drag the UI controls or blocks and design the logic of their own smart contracts. The code associated with the corresponding controls appears in the code-editor box which is marked by green boxes. After the contract designing is complete, the contract program can be saved as a ".sol" file. The following figure details the views of blocks which are marked by red box in Fig. 11.

Fig. 12 presents a snippet of the system. It shows the draggable block of "Basic Component". As shown in the yellow box, it consists of the basic contract code like "SafeMath" and the standard interface code like "ERC20".

Fig. 13 presents the draggable block of "function". It consists of a set of basic function methods such as "transfer" function, "transferFrom" function, "approve" function and so on.

Above blocks of UI controls wire up the basic function codes which are generated by improved Char-RNN model. The codes that appear in the code-editor box are exactly the generated codes in Part B by improved Char-RNN model.

2) CASE STUDY OF SMART CONTRACT DESIGNING

This section presents the process of smart contract designing by the visualized page editor. And a case study of contract for trading contract named "MyContract" is designed.

The designed contract "MyContract" defines the information of food trading in details, such as the total trading value, a series of operations for transactions and the agreement with other traders. Considering the operations for food transactions between traders, the designed contract includes many basic functions. Among of these basic functions, for instance, "transfer" function is add into the designed contract by users for transferring money during trading, the "balanceOf" function is added for querying the balance of traders, the "SafeMath" contract is added for safe math operations during the transaction. Eventually, the designed contract "MyContract" for food trading need to be executed automatically and successfully on the Ethereum blockchain network. Thus, this paper adopts the existing framework [18] named "Credit Evaluation System Based on Blockchain" to make sure the designed contract useable. This framework was just proposed for executing food trading and supervising the traders in food supply chain by smart contracts. The process of contract designing by users is detailed below (Fig. 14 and Fig. 15).

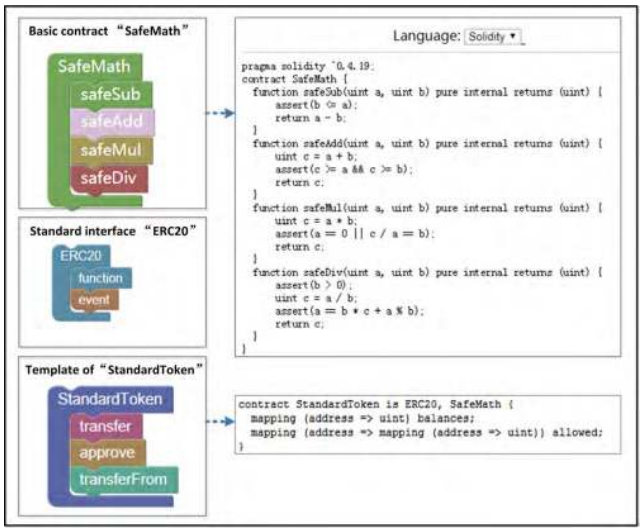

FIGURE 14. The design process and result of basic functions.

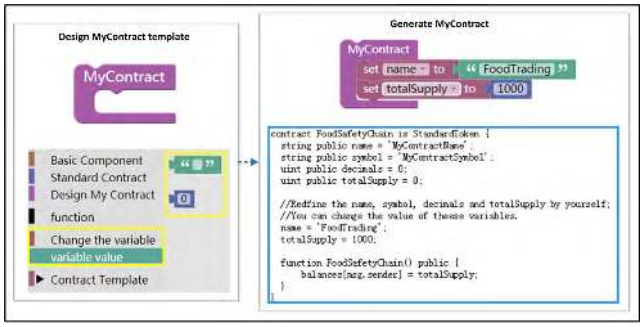

FIGURE 15. The design process and result of "MyContract".

Step one, user drag the UI control named ''SafeMath'' and ''ERC20'' on the editor. The ''SafeMath'' control links to the basic contract ''SafeMath'' generated by improved Char-RNN model. And the ''ERC20'' control links to the generated codes of standard interface ''ERC20''. Fig. 14 gives the code snippet of ''SafeMath''. It offers the unified arithmetic function (such as add, subtract, multiply and divide) for traders to perform the trading more normative.

Step two, a specialized template of ''StandardToken'' is given at the bottom of the Fig. 14. This standard contract ''StandardToken'' inherits from the basic contract ''SafeMath'' and the standard interface ''ERC20''. Its member functions use encapsulated calls to the base class members to enforce type safety. For example, considering the process of food trading, traders need to transfer money between their accounts. To achieve this operation, the user needs to drag the UI control about ''transfer'' function and drop it in the control of ''StandardToken''. At the same time, the code of ''transfer'' function appears in the code-editor box on the right of the page editor. As showed just before Fig. 7, the block of ''transfer'' function and the appeared code are marked by blue boxes. In particular, the appeared code is the basic function method of ''transfer'' generated by improved Char-RNN. It is embedded into the corresponding block and wires up to the graphical block ''transfer''.

In addition to the ''transfer'' function, users also can add other basic function methods. Similarly, UI controls of ''functions'' (Fig. 13) offers a set of basic function methods used for trading. For example, the controls of ''approve'', ''transferFrom'' and other functions all can be added into the ''StandardToken'' to meet the real needs of trading. And they link to the respective codes of basic function methods that are generated by improved Char-RNN.

Step three, after the standard contract ''StandardToken'' is designed, user can realize the personalized contract ''MyContract'' through the inheritance of ''StandardToken''. The instance of designed data all pass into ''MyContract''. Then user can add new features or redefine the existing features of contract by controls. For example, as shown in Fig. 15, the contract's name can be redefined as ''FoodTrading'' by corresponding UI controls. It shows a direct statement about the purpose of the trading contract. The program of ''MyContract'' is updated in the code-editor box. So far, the introductions of all of the functional components and designing steps of food trading contract ''MyContract'' are completed.

The program of ''MyContract'' appeared in the code-editor box can be saved as the file named ''MyContract.sol''. For validating the usability of the designed smart contracts, food trading contract ''MyContract.sol'' is deployed and complied in the existing Credit Evaluation System. Similarly, the system in this paper also can generate the evaluation contract which can be adopted in the Credit Evaluation System. The result presents that designed contract is compiled and executed successfully and it achieves a good integration with the existing Credit Evaluation System.

VI. CONCLUSIONS

This paper provides a user-defined smart contract designing system. It proposes a visual programming platform for users so that they can interact with the platform and design their smart contracts through the visualized interface. Users can obtain the smart contract codes written by Solidity language based on their definition and requests. In comparison with existing methods, the system offers a unique approach to code auto-generation. The system adopts improved Char-RNN model to generate the unified and modular basic function codes of trading contracts and it embeds the generated codes into the draggable UI blocks. One the one hand, the system strengthens the designability for non-developer users like domain experts and other contract participants who are not good at performing code work. The system offers the normalize templates of smart contracts for trading. Thus, these normal users of smart contracts don't need to understand the specialized programming language (Solidity). Users just need to design their smart contracts and meet their business requirements by dragging the UI controls. When users drag

the function blocks together, the corresponding codes appear in the visualized page editor. On the other hand, compared with the manual programming, the system implements the automatic coding. The system also provides a more efficient, fast and easy way than traditional ways to design smart contracts for professional developers. It greatly reduces the difficulty in programming and save the time for contract programming.

By validating the usability of the system, the designed contracts by users have been done the usability tests on the Credit Evaluation System. However, there are several limitations in this article. Firstly, this paper focused heavily on the studies for generating the trading contract codes. Thus, the trying for code auto-generation of more domains needs to be explored in further research. Secondly, this paper just validates the availability and usability of the generated smart contracts. It adopts more commonly used and effective algorithms to implement the automatic classification and coding of smart contracts. Thus, for more exact and effective automatic generation, there is a lot of room to improve the current algorithms used in the system.

REFERENCES

[1] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "SoK: Research perspectives and challenges for bitcoin and cryptocurrencies," in *Proc. Secur. Privacy*, May 2015, pp. 104–121.
[2] Y. Yuan and F.-Y. Wang, "Blockchain and cryptocurrencies: Model, techniques, and applications," *IEEE Trans. Syst. Man, Cybern., Syst.*, vol. 48, no. 9, pp. 1421–1428, Sep. 2018.
[3] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. Secur. Privacy*, May 2016, pp. 839–858.
[4] C. K. Frantz and M. Nowostawski, "From institutions to code: Towards automated generation of smart contracts," in *Proc. IEEE Int. Workshops Found. Appl. Self Syst.*, Sep. 2016, pp. 210–215.
[5] M. Bartoletti and L. Pompianu, "An empirical analysis of smart contracts: Platforms, applications, and design patterns," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, 2017, pp. 494–509.
[6] A. Bogner, M. Chanson, and A. Meeuw, "A decentralised sharing app running a smart contract on the ethereum blockchain," in *Proc. 6th Int. Conf. Internet Things*, 2016, pp. 177–178.
[7] G. Peters, E. Panayi, and A. Chapelle, *Trends in Crypto-Currencies and Blockchain Technologies: A Monetary Theory and Regulation Perspective*, vol. 3. New York, NY, USA: Social Science Electronic Publishing, vol. 2015.
[8] R. Adams, G. Parry, P. Godsiff, and P. Ward, "The future of money and further applications of the blockchain," *Strategic Change.*, vol. 26, no. 5, pp. 417–422, 2017.
[9] M. Al-Bassam, "SCPKI: A smart contract-based PKI and identity system," in *Proc. ACM Workshop Blockchain, Cryptocurrencies Contracts (BCC)*, 2017, pp. 35–40.
[10] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 254–269.
[11] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 7, pp. 1366–1385, Jul. 2018.
[12] N. Atzei, M. Bartoletti, T. Cimoli, "A survey of attacks on Ethereum smart contracts," in *Proc. Int. Conf. Princ. Secur. Trust*, 2017, pp. 1–24.
[13] M. Alharby and A. van Moorsel, "Blockchain based smart contracts: A systematic mapping study," in *Proc. Int. Conf. Artif. Intell. Soft Comput.*, 2017, pp. 125–140.
[14] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "SmartCheck: Static analysis of Ethereum smart contracts," in *Proc. IEEE/ACM 1st Int. Workshop Emerg. Trends Softw. Eng. Blockchain (WETSEB)*, May/Jun. 2018, pp. 9–16.
[15] B. K. Mohanta, S. S. Panda, and D. Jena, "An overview of smart contract and use cases in blockchain technology," in *Proc. 9th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, Jul. 2018, pp. 1–4.
[16] W. Ling, I. Trancoso, C. Dyer, and A. W. Black, "Character-based neural machine translation," 2015, *arXiv:1511.04586*. [Online]. Available: https://arxiv.org/abs/1511.04586
[17] A. Mikami, "Long short-term memory recurrent neural network architectures for generating music and Japanese lyrics," Ph.D. dissertation, Boston College, Newton, MA, USA, 2016.
[18] D. Mao, F. Wang, Z. Hao, and H. Li, "Credit evaluation system based on blockchain for multiple stakeholders in the food supply chain," in *Int. J. Environ. Res. Public Health*, vol. 15, no. 8, p. 1627, 2018.
[19] P. Moreno-Sanchez, M. B. Zafar, and A. Kate, "Listening to whispers of ripple: Linking wallets and deanonymizing transactions in the ripple network," in *Proc. Privacy Enhanching Technol.*, 2016, pp. 436–453.
[20] A. B. Ayed and M. B. Belhajji, "The blockchain technology: Applications and threats," *Int. J. Hyperconnect. Internet Things.*, vol. 1, no. 2, pp. 1–11, 2017.
[21] S. Omohundro, "Cryptocurrencies, smart contracts, and artificial intelligence," *AI Matters*, vol. 1, no. 2, pp. 19–21, 2014.
[22] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.
[23] K. Gkillas and P. Katsiampa, "An application of extreme value theory to cryptocurrencies," *Econ. Lett.*, vol. 164, pp. 109–111, Mar. 2018.
[24] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," Cryptology ePrint Archive, Tech. Rep. 2019/191, 2019. [Online]. Available: https://eprint.iacr.org/2019/191
[25] D. Mohanty, "Deploying smart contracts," in *Ethereum for Architects and Developers*. Berkeley, CA, USA: Apress, 2018, pp. 105–138.
[26] H. C. Wu, R. W. P. Luk, K. F. Wong, and K. L. Kwok, "Interpreting TF-IDF term weights as making relevance decisions," *ACM Trans. Inf. Syst.*, vol. 26, no. 3, pp. 55–59, 2008.
[27] P. Niu and D. G. Huang, "TF-IDF and rules based automatic extraction of Chinese keywords," *J. Chin. Comput. Syst.*, vol. 37, no. 4, pp. 711–715, 2016.
[28] T. Shouzhong and H. Minlie, "Mining microblog user interests based on TextRank with TF-IDF factor," *J. China Univ. Posts Telecommun.*, vol. 23, no. 5, pp. 44–50, 2016.
[29] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.
[30] A. Graves. (2012). *Supervised Sequence Labelling With Recurrent Neural Networks*. [Online]. Available: http://books.google.com/books
[31] Z. C. Lipton, S. Vikram, and J. McAuley, "Capturing meaning in product reviews with character-level generative text models," 2015, *arXiv:1511.03683*. [Online]. Available: https://arxiv.org/abs/1511.03683
[32] I. Culic, A. Radovici, and L. M. Vasilescu, "Auto-generating Google Blockly visual programming elements for peripheral hardware," in *Proc. RoEduNet Int. Conf.-Netw. Educ. Res.*, Sep. 2015, pp. 94–98.

DIANHUI MAO was born in 1979. He received the degree from the Huazhong University of Science and Technology. He is currently an Associate Professor with the College of Computer and Information Engineering, Beijing Technology and Business University. He is a Distinguished Expert with the Global Blockchain Industry Research Institute, China Mobile Communications Federation. He has a wealth of experience in the application studies of blockchain and AI. In recent years, he has presided over and participated in projects including the National Social Sciences Fund Project, the Humanities and Social Science Fund Project of the Ministry of Education, and various types of enterprise customization systems. He has published over 20 papers. He has received the Leading Talent of Enterprise Innovation and Entrepreneurship Award in Jiangsu, China. He is also a member of Think Tank Committee, All-China Federation of Industry and Commerce (ACFIC).

FAN WANG was born in 1994. She received the M.S. degree in computer engineering from the College of Computer and Information Engineering, Beijing Technology and Business University. She is currently involved in the integration and the development of blockchain technology and deep learning technology. She holds one patent about blockchain. She has already published one paper for conducting research on food safety supervision and management according to blockchain and smart contracts.

YALEI WANG was born in 1993. He is currently pursuing the degree in computer engineering with the College of Computer and Information Engineering, Beijing Technology and Business University. He is currently involved in the integration and the development of blockchain technology, and deep learning technology for food safety supervision and management.

ZHIHAO HAO was born in 1997. He received the bachelor's degree in computer science and technology from the College of Computer and Information Engineering, Beijing Technology and Business University. He will pursue the master's degree with the Pattern Analysis and Machine Intelligence Group, Department of Computer and Information Science, University of Macau. He is currently involved in the applications of blockchain technology. He holds one patent about blockchain. He has already published two papers for conducting research on blockchain and smart contracts applications.