Journal of Network Intelligence ©2025 ISSN 2414-8105
Taiwan Ubiquitous Information Volume **10**, Number **3**, August **2025**

Intelligent Software Development Vulnerability Identification Based on Question-and-Answer Modeling and Attention Mechanisms

Xin Feng[1,*], Dong-Hua Zheng[2]

[1]School of Modern Information Engineering, Hunan Chemical Vocational Technology College
Zhuzhou 41200, P. R. China
fengxinxin8281@126.com

[2]Graduate School, Management and Science University
Shah Alam 40100, Selangor, Malaysia
20190257@gcc.edu.cn

*Corresponding author: Xin Feng

Received March 29, 2024, revised September 26, 2024, accepted December 25, 2024.

Abstract. With the popularization of the Internet and the wide application of software, the existence of vulnerabilities may lead to serious security threats and losses. In this context, intelligent software vulnerability identification technology has become the key to guarantee the security of software systems. Traditional recognition methods disregard the contextual information or word order in the vulnerability description, which seriously reduces the accuracy of vulnerability recognition. Focusing on the above issues, this article suggests a vulnerability identification method for intelligent software development relied on Question & Answer (Q&A) model and attention mechanism. Firstly, according to the features of intelligent software development, BERT is adopted to extract the characterization of input code vulnerability sequences, and a vulnerability Q&A model based on BERT is constructed. On this basis, the attack operation is generalized and transformed into a question, and the input vulnerability code snippets and their contexts and query descriptions are subjected to word embedding operations, replacing these input textual data information with fixed-length vectors. Then the query description and code snippet interactive representations are obtained through the two-way attention mechanism and fused as inputs to TextRCNN, which utilizes the vulnerability embedded words to output the results of vulnerability identification. The experimental outcome indicates that the recognition method designed in this article is with a low False Positive Rate (FPR), False-Negative Rate (FNR), high accuracy, precision, recall, and F1 value, and the vulnerability recognition effect is good.
Keywords: Intelligent software; Vulnerability identification; BERT; Q&A model; Attention mechanism

1. Introduction.

Computer systems are the digital cornerstone of human life, empowering the survival and development of people and businesses everywhere. However, software vulnerabilities are prevalent in cyberspace, and attackers can exploit vulnerabilities in security policies or system implementations to gain unauthorized access and jeopardize system security [1]. Advanced Persistent Threats (APTs) may even cross and combine multiple categories of vulnerabilities, thus posing a huge threat to cyberspace security, including well-known Internet companies and state organizations have suffered from vulnerability attacks [2, 3, 4]. However, due to the constraints of the developers' own ability, experience, and the current level of cybersecurity technology, it is very difficult to avoid some defects in the design and realization of the project. Moreover, the open-source code allows these flaws to spread to a wider range. Application vulnerability identification is an effective attack mitigation solution before vulnerabilities are discovered or even deployed [5, 6]. Therefore, software vulnerability identification is a necessary and far-reaching task for both software developers and users. In this paper, a new vulnerability identification method is proposed, which combines question-answering model and attention mechanism to better understand and deal with the context information and word order in vulnerability description. The method of this study is not only innovative in theory, but also has high application potential in the actual software development environment.

1.1. **Related work.** Satyanarayana and Sekhar [7] suggested a static analysis tool relied on intermediate representations for identifying vulnerabilities in intelligent software, but it runs slowly. Goseva-Popstojanova and Perhinschi [8] improved scalability by directly inspecting the code functions, rather than the entire program, in order thereby identifying vulnerabilities in the software. Shah et al. [9] et al. designed an automated testing framework to identify multiple vulnerability types in intelligent software. Zhang et al. [10] et al. used a hybrid approach of static and dynamic analysis for identifying vulnerabilities in Android applications. Wang [11] analyzed the dependencies between software components to identify the intelligent software development process Chowdhury and Zulkernine [12] investigated software vulnerability indicators to predict the presence of vulnerabilities in software systems. Rehman and Mustafa [13] created a database of vulnerable software code by massively mapping the relationship between CVEs and GitHub commits and labeled suspected vulnerabilities with a Support Vector Machine SVM classifier, but the accuracy of identification was not high. Traditional machine learning vulnerability identification methods require in-depth understanding of the characteristics of code vulnerabilities, and it is difficult to fully capture all vulnerabilities by manual means. Thanks to the rapid growth of artificial intelligence technology, scholars have found that methods such as the use of deep learning and natural language processing are more advantageous in mining the deep characteristics of vulnerabilities. Huang et al. [14] use deep learning to identify smart software vulnerabilities, and can detect four types of vulnerabilities related to arrays, arithmetic expressions, pointers, and function calls, but the identification efficiency is not high. Sun et al. [15] describe the source code based on the BERT model in a question-and-answer style, and determine whether there is a specific type of vulnerability in software. Chen et al. [16] extract the control and data dependency features between the statements of the vulnerable code, and use a converter-based model to identify smart software vulnerabilities, but ignores the code text context information. Wang et al. [17] represent the text of software code in word vectors, and use a converter-based model to identify smart software vulnerabilities. Li et al. [18] obtain the textual semantic information of software code through BiGRU, and then use LSTM to enhance the learning of textual local information to identify vulnerabilities. Wu et al. [19] proposed a software code vulnerability identification method using the attention mechanism to deal with the issue of loss of relevant information brought about by vector aggregation, and enhance the efficiency and accuracy of identification. Sun et al. [20] used the BERT model to model the similarity of question-answer pairs and combined it with the RNN deep integration model for intelligent software vulnerability identification. We choose BERT model because it is excellent in understanding context and word order, which is very important in the accuracy of vulnerability description.

1.2. **Contribution.** The intelligent software vulnerability identification methods suggested by the above research scholars neglect the contextual information or word order in the vulnerability description, which reduces the accuracy of vulnerability identification. To deal with this problem, this article designs an intelligent software development vulnerability identification method relied on Q&A model and attention mechanism. Firstly, using the information domain stored in the Q&A, BERT is used to extract the characterization of the input vulnerability code sequence, and the extracted characterization is fused with the encoder and decoder of each layer in the Neural Machine Translation (NMT) model to construct a BERT-based vulnerability Q&A model. Secondly, word embedding is performed on the vulnerability code fragments and context information, and the semantic feature matrices of code fragments, code fragment context and query description are extracted from the word embedding process by self-attention network, and then the interactive representations of query description and code fragments are obtained by the bi-directional attention mechanism and fused together. Finally, the software vulnerability classification and identification model are constructed based on TextRCNN, and the vulnerability classification is performed by using vulnerability embedding words to classify the vulnerabilities. Experimental outcome indicates that the designed method has better recognition performance and can be better applied to the field of intelligent software development recognition.

2. **Theoretical analysis.**

2.1. **Pre-trained language model BERT.**

BERT [21] is a pre-trained language model that uses the Transformer architecture [22] and learns a deeply bidirectional language representation through pre-training. Unlike previous language models, BERT can learn a generalized language representation by pre-training on a large-scale unlabeled corpus, which can then be fine-tuned to adapt to specific downstream tasks for example text categorization, Q&A, etc.

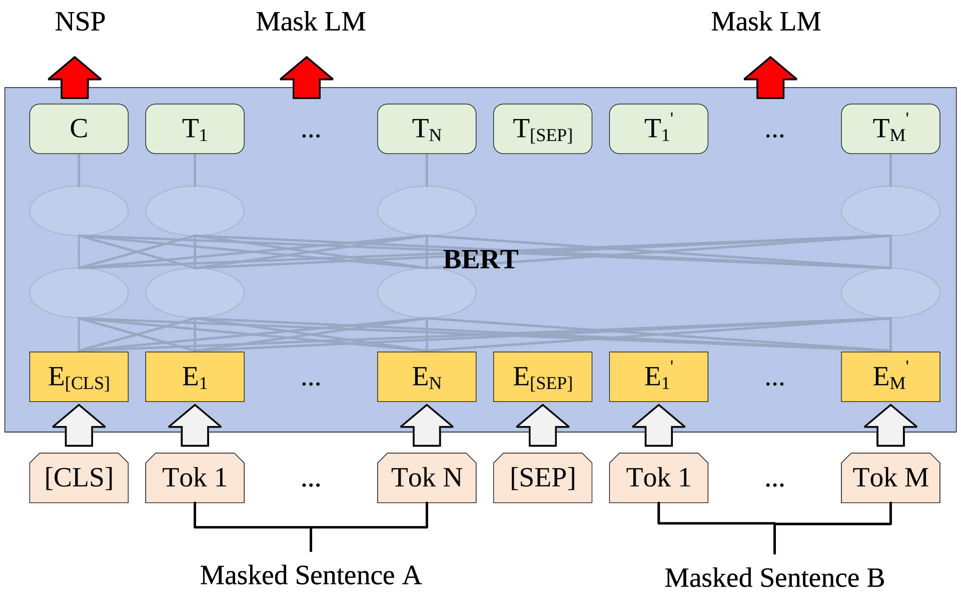

Figure 1. The model structure of BERT

The pre-training phase of BERT consists of two tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). The structure of the BERT model is shown in Figure 1. The MLM task learns the language representation by randomly masking some words in the input text so that the model learns to forecast the probability dispersion of the masked words. The purpose of this is to allow the model to study more about the structure and semantics of the language. The NSP task allows the model to determine the relationship between two given sentences. In this task, given a sentence A and a sentence B, the model needs to determine whether sentence B is a reasonable successor to A, so that the model can study the logical and semantic relationships between the sentences.

2.2. **Attention mechanism.** Attention mechanisms were firstly applied to machine translation task [23], in the aspect of text input, some important words in the utterance input account for a larger proportion of the decision-making on the utterance, but these key parts often do not get more attention from the model, so it is necessary to use the Attention Mechanisms to pay attention to the key parts, and to fuzzy and ignore other irrelevant regions that have less influence or more noise on the classification results, which can effectively enhance the accuracy of the model [24]. The essence of the attention function is to weight the mapping of a query key to a set of key-value pairs, and finally obtain the attention weights. Figure 2 illustrates the three computational stages of the attention mechanism.

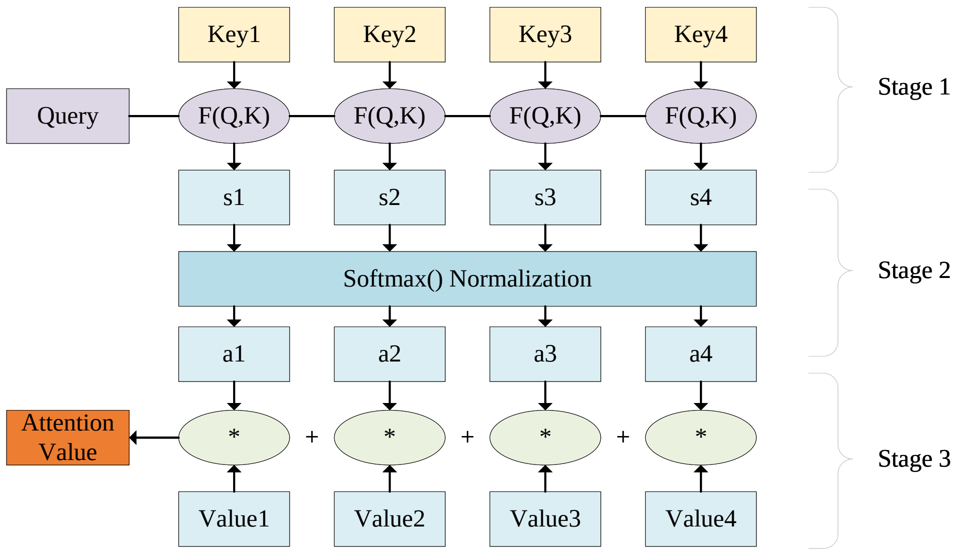

Figure 2. Attention mechanism

In the first stage, the correlation weights between the query (Q) and each key value (K) are first calculated. The common calculation methods are dot product similarity, Cosine similarity and MLP similarity as indicated in Equation (1), Equation (2) and Equation (3) respectively.

$$S_i = F(Q, K) = Q \cdot K \tag{1}$$

$$S_i = F(Q, K_i) = \frac{Q \cdot K_i}{\|Q\| \cdot \|K_i\|} \tag{2}$$

$$S_i = F(Q, K_i) = MLP(Q \cdot K_i) \tag{3}$$

In the second stage, the results of the first stage are normalized by softmax function to obtain the weight coefficients $\alpha_i = softmax(S_i) = \frac{e^{sim}}{\sum_{j=1}^{t} e^{sim}}$.

In the third stage, the attention value $att = \sum_{i=1}^{t} \alpha_i * Value_i$ is obtained by weighting and summing each Value using the weighting coefficients α_i from the second stage.

3. **BERT-based vulnerability Q&A model for intelligent software development.** Traditional Q&A models are mostly realized by question similarity matching based on Q&A databases, nevertheless, this method is not efficient and needs high labeling cost to build up the training dataset. Intending tot, the above issues, this article adopts two parts, encoder and decoder, to form a downstream neural network machine translation model. The encoder part mainly takes the output of the BERT model as the input of the attention layer of the coder, and outputs a series of hidden units in a loop. The decoder part mainly takes the output of BERT model as the input to the attention level of the decoder, and takes the hidden units of the encoder as the input to the other attention layer of the decoder, and then outputs the corresponding output units of each hidden unit in a loop. The steps in detail are as bellow.

Step 1: Let the smart software vulnerability code fragment be X, and given any sentence $x \in X$, input it into the BERT model to obtain the characterization output $G_B = BERT(x)$, where $G_{B,i} \in G_B$ is the characterization of the i-th wordpiece in x. Step 2: Let G_E^k be the obscured representation of the k-th layer of the encoder, and G'_E be the word embedding of the vulnerability code sequence x. For any $i \in [k_x]$, g_i^k is the i-th element of G_E^k, and k_x is the length of the sentence x. For layer k, Equation (4) can be obtained.

$$\tilde{g}_i^k = \frac{1}{2}\left(att_s(g_i^{k-1}, g_E^{k-1}, g_E^{k-1}) + att_B(g_i^{k-1}, G_B, G_B)\right), \forall i \in [k_x] \tag{4}$$

where att_s and att_B represent two attention modules with different parameters, and their calculations are indicated in Equation (5) and Equation (6), respectively.

$$att_s(g_i^{k-1}, g_E^{k-1}, g_E^{k-1}) = \text{softmax}\left(\frac{g_i^{k-1}(g_E^{k-1})^T}{\sqrt{d_k}}\right) g_E^{k-1} \tag{5}$$

$$att_B(g_i^{k-1}, G_B, G_B) = \text{softmax}\left(\frac{g_i^{k-1}(G_B)^T}{\sqrt{d_k}}\right) G_B \tag{6}$$

where softmax denotes the activation function and d_k is the dimension of the input vector of the smart software vulnerability code.

Subsequently, each $\tilde{g}_i^k$ is further processed using a feed-forward network (FFN) [25], which is represented as follows.

$$FFN(x) = V_2 \max(V_1 x + b_1, 0) + b_2 \tag{7}$$

where V_1 and V_2 denote the weights of the first and second layers of the encoder, respectively; b_1 and b_2 denote the bias terms of the first and second layers of the encoder, respectively. FFN is a key component in the BERT model, which is located at the end of each Transformer encoder layer. Each encoder layer of BERT model consists of Self-Attention Mechanism and FFN. The self-attention mechanism enables the model to capture the dependencies between different positions in the input sequence, and FFN further transforms and abstracts these dependencies. Here the output value of the k-th layer will be obtained as Equation (8).

$$G_E^k = (FFN(\tilde{g}_1^k), ..., FFN(\tilde{g}_{k_x}^k)) \tag{8}$$

The encoder will eventually output G_E^k from the last layer, where K is the total amount of layers of the encoder.

Step 3: Let S_d^k be the k-th hidden state of the decoder before time step t, and have $S_d^k = (s_t^k, \cdots, s_{t-1}^k)$. It is worth noting that s_1^0 is a special marker representing the beginning of the software exploit code sequence, and s_t^0 is the word embedding of the predicted exploit code word at time step $t-1$. For the k-th layer, firstly, calculate the self-attention s_t^k of the obscured state of the $k-1$-th level before the time step t, secondly, calculate the self-attention s_t^k with G_B and G_E^k respectively, and sum the outputs of these two attention layers to get s_t^k. Finally, input s_t^k into the nonlinear transformation layer FFN to get the s_t^k of each layer, i.e., $s_t^k = FFN(s_t^k)$, and then get the t-th predicted vulnerability code word y_t by softmax. and the computation equations of s_t^k and $\tilde{s}_t^k$ are as indicated in Equation (9) and Equation (10) respectively.

$$s_t^k = att_s(s_t^{k-1}, S_d^{k-1}, S_d^{k-1}) \tag{9}$$

$$\tilde{s}_t^k = \frac{1}{2}(att_B(s_t^k, H_B, H_B)) + att_E(s_t^k, G_E^K, G_E^K)) \tag{10}$$

where att_s, att_B and att_E represent the self-attention model, the BERT-decoder attention model and the coder-decoder attention model, respectively. The s_t^k of each layer can be obtained through s_t^k and $\tilde{s}_t^k$. The final s_t^k is mapped to a linear transformation and softmaxed to obtain the t-th predicted word $\hat{y}_t$. The decoding process will continue until the end-of-sentence identifier is encountered.

4. Intelligent software development vulnerability identification based on Q&A modeling and attention mechanisms.

4.1. Software vulnerability code word embedding based on BERT Q&A modeling. Intending to the issue that existing identification methods do not fully consider the contextual semantic features, leading to low identification accuracy, this article designs an intelligent software development vulnerability identification method relied on Q&A model and attention mechanism. Firstly, based on the BERT Q&A model, word embedding is performed on the vulnerability code fragments and context information, secondly, the semantic feature matrix of code fragments, code fragment context and query description after word embedding is extracted by the self-attention network, and then the interactive representations of the query description and code fragment are obtained through the bidirectional attention mechanism and fused, and finally, the software vulnerability classification and recognition model is constructed based on TextRCNN, and the vulnerability embedding word embedding is utilized to identify vulnerabilities. The overall process is indicated in Figure 3.

The integration of attention mechanism and BERT model is realized in BERT's self-attention layers, which are the core components of BERT architecture. In BERT model, the attention mechanism allows the model to consider other words in the input sequence when processing a word, thus capturing the contextual relationship between words. This mechanism is very important for understanding complex language structures and semantic information. In this article, based on the above BERT Q&A model, the attack operation is generalized into a question, and the input vulnerability code snippet and its context and query description are subjected to word embedding operation, which replaces these input textual data information with fixed-length vectors. For a given vulnerability code segment $D_i = d_{i1}, ..., d_{|d_i|}$ and its textual contexts $T_i = t_{i1}, ..., t_{|T_i|}$, $T_{i+1} = (t_{i+1})_1, ..., (t_{i+1})_{|T_{i+1}|}$ and corresponding query description $Q = q_1, ..., q_{|q|}$, the designed question and software code vulnerability descriptions are organized into a suitable sequential shape. The output answer consists of sequential words in the vulnerability description.

BERT introduces a start vector $S \in \mathbb{R}^c$ and an end vector $E \in \mathbb{R}^c$. Use P_i to compute the dot product between the context and the start vector S, and then apply the Softmax function to the dot product $S \cdot T_i$ to obtain the probability that the i-th word is the start of the answer. The probability that the j-th word is the end of the answer is calculated using P_j. The score of a candidate answer is defined as $S_{i,j} = S \cdot T_i + E \cdot T_j$ and is used as the answer when the maximum score span of $j \geq i$ is reached.

Finally, the corresponding word embedding of each word in the pre-trained question query text sequence of BERT is used to construct its corresponding sequence encoding vector, and the embedding vector of the vulnerability code words is obtained as follows.

$$d_{il}^{e} = E_D \cdot d_{il} (l \in [1, |D_i|]) \tag{11}$$

where $E_D \in \mathbb{R}^{c \times W_D}$ is the matrix of vulnerability code word embeddings pre-trained by BERT and c is the dimension of the word embeddings.

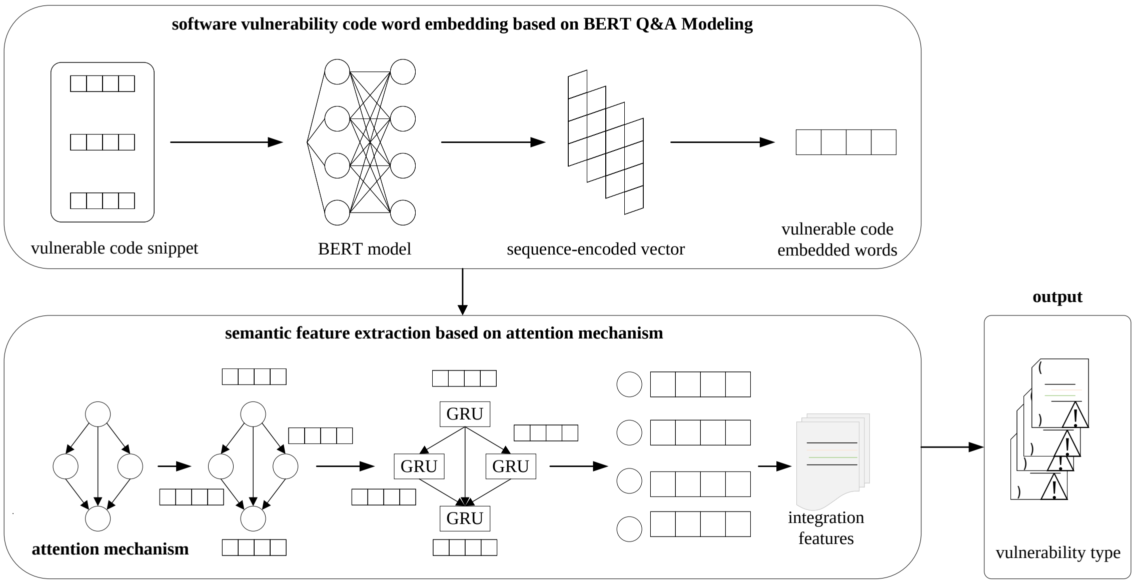
software vulnerability code word embedding based on BERT Q&A Modeling
vulnerable code snippet
BERT model
sequence-encoded vector
vulnerable code embedded words
semantic feature extraction based on attention mechanism
attention mechanism
GRU
GRU
GRU
GRU
integration features
output
vulnerability type

Figure 3. The entire process of the suggested method

4.2. **Semantic feature extraction based on attention mechanism.** After mapping the sequential words of the vulnerability code snippets and their contexts and query descriptions to word embedding vector representations, the underlying fine-grained features of the code snippets and their contexts and query descriptions can be encoded by summarizing the embedding vector representations of the words in the forward and backward directions of the text sequences through self-attentive networks based on self-attentive mechanisms. For the code fragment $D_i^e = C_1^e, ..., D_{|D_i|}^e$ after word embedding, after constructing word embeddings for each word using the above steps, each word has the same representation dimension c, and then D_i^e is converted into a query vector $Q_{D_i} \in \mathbb{R}^{|D_i| \times c}$, a key vector $K_{D_i} \in \mathbb{R}^{|D_i| \times c}$, and a value vector $W_v \in \mathbb{R}^{|D_i| \times c}$, where $Q_{C_i} = D_i^e \cdot V_Q^T$, $K_{C_i} = D_i^e \cdot V_K^T$, and $V_{C_i} = D_i^e \cdot V_w^T$, by three different weights $V_Q \in \mathbb{R}^{c \times c}$, $V_K \in \mathbb{R}^{c \times c}$, and $V_v \in \mathbb{R}^{c \times c}$.

After completing the above steps, the semantic features of the software vulnerability code sequence are captured using the self-attention mechanism, which is computed as bellow.

$$context_{D_i} = att(Q_{D_i} \cdot K_{D_i}) \cdot V_{D_i} \tag{12}$$

where $att(\cdot)$ is a scaled dot product attention model.

$$att(Q_{D_i} \cdot K_{D_i}) = softmax((Q_{D_i} \cdot K_{D_i}^T)/\sqrt{c}) \tag{13}$$

where $\sqrt{c}$ is a temperature factor that prevents the gradient of the model from disappearing during the training phase, and it has the same representation dimension c. $softmax(\cdot)$ is a normalization function, which is the matrix used to obtain the attention, and after completing these steps, by applying the $context_{D_i}$-vector to a simple fully-connected feed-forward neural network in the position direction, we can obtain the final output $w_{D_i} \in \mathbb{R}^{|D_i| \times c}$, which is indicated in the following equation.

$$w_{D_i} = \mathrm{Relu}(context_{D_i} \cdot V_1 + b_1) \cdot V_2 + b_2 \tag{14}$$

where $V_1 \in \mathbb{R}^{c \times f},\ V_2 \in \mathbb{R}^{f \times c},\ b_1 \in \mathbb{R}^f,\ b_2 \in \mathbb{R}^c$, where f is the dimension of the obscured level in the feedforward neural network and $Relu(\cdot)$ is the activation function.

Based on the sequence information encoded in the software vulnerability code fragment, for the query description sequence $Q^e = q^e_1, ..., q^e_{|Q|}$ that has been word-embedded, Q^e is first transformed into a query vector $Q_Q \in \mathbb{R}^{|Q| \times c}$, a key vector $K_Q \in \mathbb{R}^{|Q| \times c}$, and a value vector $V_Q \in \mathbb{R}^{|Q| \times c}$, and then a self-attention mechanism is utilized to capture the semantic characteristics of the query description sequence. Finally, the final output $w_Q \in \mathbb{R}^{|Q| \times c}$ is obtained by applying the output of the self-attentive network vectors to a simple feed-forward fully connected layer.

$$context_Q = att(Q_Q \cdot K_Q) \cdot V_Q \tag{15}$$

$$w_Q = \mathrm{Relu}(context_Q \cdot V_1 + b_1) \cdot V_2 + b_2 \tag{16}$$

Similarly, for contexts T^e_i and T^e_{i+1} after word embedding of the vulnerability code fragment, the semantic information of the context sequence is captured using the self-attention mechanism. The final outputs $w_{T_i} \in \mathbb{R}^{|T_i| \times c}$ and $w_{T_{i+1}} \in \mathbb{R}^{|T_{i+1}| \times c}$ are obtained.

The software vulnerability code snippets and their query descriptions can be obtained after passing through the self-attention network layer with their semantic feature matrices w_{D_i} and w_Q. The interactive attention matrix $G \in \mathbb{R}^{|D_i| \times |Q|}$ is computed by introducing the parameter matrix $U \in \mathbb{R}^{c \times c}$ learned by the neural network, which is given by the following equation.

$$G = \tanh(w_{D_i} U w_Q) \tag{17}$$

An average pooling operation is performed along rows and columns on G to obtain semantic vectors $f^D \in \mathbb{R}^{|D_i|}$ and $f^Q \in \mathbb{R}^{|Q|}$ for vulnerability codes and query descriptions, as indicated below.

$$f^D = [f^D_1, ..., f^D_{|D_i|}] \tag{18}$$

$$f^Q = [f^Q_1, ..., f^Q_{|Q|}] \tag{19}$$

After that, softmax function is used on semantic vectors f^D and f^Q to generate attention vectors $a^D \in \mathbb{R}^{|D_i|}$ and $a^Q \in \mathbb{R}^{|Q|}$ for vulnerability code fragments and query descriptions. A dot product is implemented between the feature matrices w_{D_i}, w_Q and the attention vectors a^D, a^Q to generate the common attention representations $r^D, r^Q \in \mathbb{R}^c$ for the vulnerability code and query description, respectively. Similarly the context feature matrices w_{T_i} and $w_{T_{i+1}}$ of the exploit code fragment are average pooled to obtain the final representations of the text context $f^{\tau}_{T_i} \in \mathbb{R}^c$ and $f^{\tau}_{T_{i+1}} \in \mathbb{R}^c$. f^{τ}_i, r^D, r^Q and $f^{\tau}_{T_{i+1}}$ are treated as four words of a sequence I and fused using Bidirectional Gated Recurrent Units (BiGRUs) to obtain the final fused feature $H_i \in \mathbb{R}^{2u}$, where H denotes the hidden layer state of the sequence and u denotes the length of the hidden layer of each unidirectional GRU.

4.3. **Classification identification output.** Since the RNN model is able to excellently capture the long-range dependencies of vulnerability descriptions. Thus, TextRCNN is chosen as the classifier for the identification stage of this paper [25]. The final fused feature H_i is inputted into Bi-LSTM to obtain the obscured state vector. The loop framework of Bi-LSTM can get the left side of the word when the text is scanned forward, and the right side of the word when it is scanned backward, and finally understand the meaning of the word by the left and right information of the word and itself. The characteristic representation of word i is learned as indicated in Equation (20).

$$X_i = [C_l(H_i); e(H_i); C_r(H_i)] \tag{20}$$

where $C_l(v_i)$ denotes the information to the left of word i and $C_r(v_i)$ denotes the information to the right of word i.

H_i is fed into the fully connected level, and the maximum pooling level is used to obtain the text feature representation to seek the most significant potential semantic information in the vulnerability description, and the text representation is computed as shown in Equation (21). Then all the pooled feature values are stitched together and the output of the model is calculated using a linear function as in Equation (22). Finally, Equation (23) is applied to calculate the probability of each type to identify the output vulnerability types.

$$Y = \max_{1 \le s \le n} v_i = \max_{1 \le s \le n} (\tanh(V_1 X_i + b_1)) \tag{21}$$

$$O = V_2 v + b_2 \tag{22}$$

$$P_i = \frac{\exp(O)}{\sum\limits_{l=1}^{n} \exp(O_l)} \tag{23}$$

5. Performance testing and analysis.

5.1. Comparative performance analysis of different identification methods.

For the purpose of estimating the performance of intelligent software development vulnerability identification methods based on question-answer models and attention mechanisms, this article conducts comparative experiments on the identification of intelligent software vulnerabilities on the SARD dataset [26], which is a software assurance reference dataset from the National Institute of Standards and Technology (NIST, USA). SARD dataset contains 6 vulnerability types and 92328 vulnerability samples. The vulnerability sample dataset is separated into training, validation, and testing sets by 6:3:1. SCVD [19], ASVC [20] and the recognition method designed in this article, OURS, are trained and tested respectively.

To test the accuracy of recognition more effectively, this article adopts ten-fold cross-validation, where the dataset is split into 10 parts, and nine parts of the data are taken in turn as training and one part of the data as testing. The dimension of each code text feature vector is 64. The size of the feed-forward neural network input data batch samples is set to 64. In this paper, the learning dropout rate is set to 0.5. In order to calculate the gradient descent of the model more efficiently, the ADAM [27] algorithm is used in this paper, in which the studying rate is set to 0.001.

To evaluate the performance of the recognition methods designed in this paper, comparative experiments on the test set of SCVD, ASVC and OURS methods will be conducted using the False Positive Rate (FPR), False-Negative Rate (FNR) [28], Accuracy (Acc), precision Prec, recall Rec, and reconciled mean F1 [29] as the measures of recognition performance. Table 1 demonstrates the results of comparison experiments on the comprehensive performance of vulnerability recognition of OURS method and SCVD and ASVC methods.

The proposed method, OURS, far exceeds the SCVD and ASVC methods in all categories. In the comparison of these three methods, SCVD has the highest false positive rate. Second, when identifying complex program codes of intelligent software, some security codes of the SCVD method may match the rule descriptions and be wrongly identified as vulnerabilities, thus the accuracy rate is not high, which requires manual confirmation

Table 1. Vulnerability identification performance comparison of different methods

Method	FPR	FNR	Prec	Rec	F1
SCVD	0.092	0.107	0.793	0.849	0.820
ASVC	0.061	0.069	0.871	0.836	0.853
OURS	0.027	0.031	0.942	0.968	0.955

and cannot achieve high usability in reality. The recall rate of the ASVC method is not high, which may be due to the fact that the vulnerability data is not clearly defined or the vulnerabilities are hidden in complex codes, although the BERT model is used to model the similarity of vulnerabilities, the fine-grained features of the vulnerability code are not extracted. From the results, the F1 score of OURS on the dataset is 0.955, which is improved by 16.46% and 11.96% compared to the SCVD method and ASVC method, respectively. In general, the OURS method shows better performance in identifying intelligent software vulnerabilities.

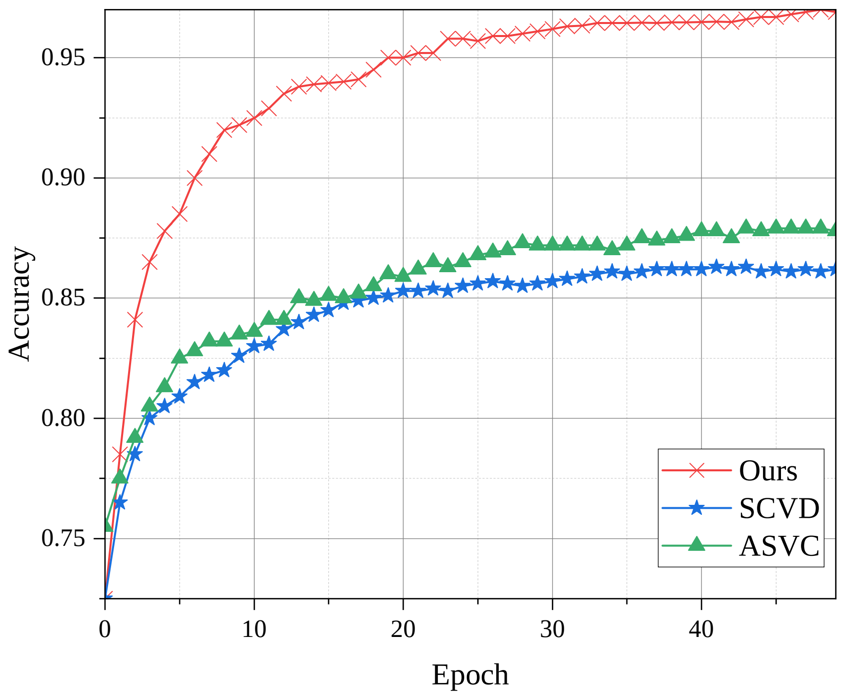

Figure 4. Comparison of accuracy of different recognition methods

The trend of the training accuracy of different vulnerability identification methods on the dataset is indicated in Figure 4. From the results in the figure, it can be seen that with the increase of the number of training iterations, the training accuracy of each model gradually increases, OURS tends to stabilize after 30 epochs of training and shows the best training accuracy of 97% on various programming language datasets, while the SCVD method has an accuracy of 85% after 35 epochs of training, and the ASVC method has an accuracy of 87% after 35 epochs of training. Because the OURS method adds the BERT model on the basis of the traditional software vulnerability identification method, and fully extracts the features of the context of the vulnerability code to improve the process of extracting the entities of the vulnerability code feature domain, and constructs the vulnerability code fragments that contain more semantic and syntactic information, it has the fastest pre-fitting speed on all kinds of datasets.

5.2. **Comparison and analysis of time expenses.** Figure 5 indicates a comparison of the average time overhead of SCVD, ASVC and OURS methods. It can be seen that the time overheads of OURS and SCVD and ASVC vulnerability identification methods are generally higher in the model training and preprocessing phases, while the time difference in the identification phase is smaller. However, because the OURS method adopts the BERT model structure and implements the extraction of vulnerability code and its context and query information features in the preprocessing stage, it relies on the deep learning model for classification in the preprocessing stage, which has a higher time overhead compared to the other methods. In the training phase, the training is performed by bidirectional loop gating units, which reduces the training time. Finally, contextual information or word order features are fully integrated in the vulnerability description of the OURS method, which significantly reduces the recognition time.

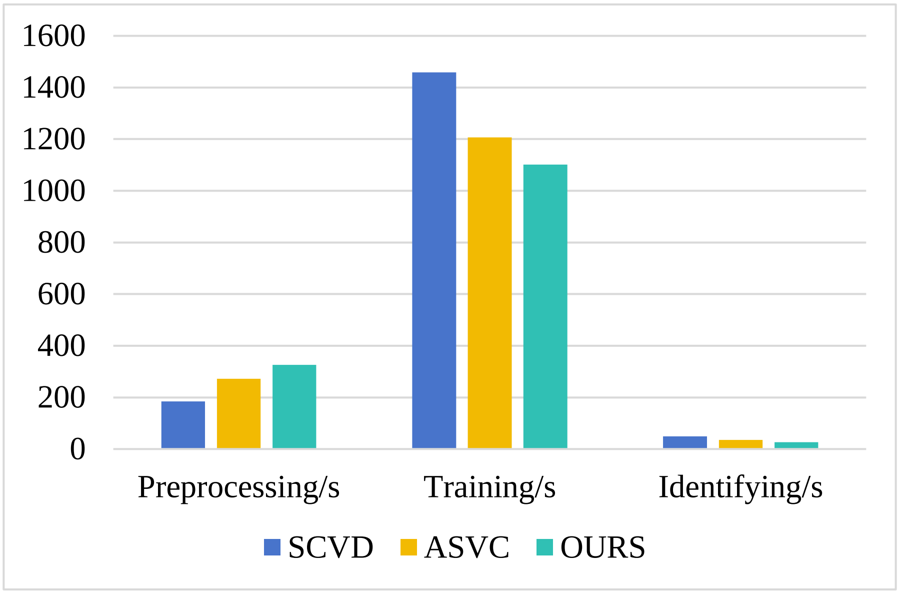

Figure 5. Comparison of time expenditure

6. **Conclusion.** Intending to the issue that existing intelligent software vulnerability identification methods disregard the contextual information or word order in the vulnerability descriptions, which reduces the accuracy of vulnerability identification, this article suggests an intelligent software development vulnerability identification method relied on the Q&A model and attention mechanism. First, adopting BERT to extract the characterization of the input vulnerability code sequences, and then fusing the extracted characterization with the encoders and decoders in each layer of the neural machine translation model to construct a BERT-based vulnerability question-answer model. On this basis, this article performs word embedding of vulnerability code fragments and context information, extract the semantic feature matrix of code fragments, code fragment context and query description after word embedding process through self-attention network, then obtain the interactive representation of query description and code fragments through the bidirectional attention mechanism, and fuse them as inputs to TextRCNN, and output the vulnerability classification results by using the vulnerability embedded words. The suggested method has better performance in FPR, FNR, accuracy rate, precision rate, recall rate and F1 value, which verifies the efficiency of the designed method.

REFERENCES

[1] V. Smyth, "Software vulnerability management: how intelligence helps reduce the risk," Network Security, vol. 2017, no. 3, pp. 10-12, 2017.

[2] S. Singh, P. K. Sharma, S. Y. Moon, D. Moon, and J. H. Park, "A comprehensive study on APT attacks and countermeasures for future networks and communications: challenges and solutions," The Journal of Supercomputing, vol. 75, pp. 4543-4574, 2019.

[3] G. Di Tizio, M. Armellini, and F. Massacci, "Software updates strategies: A quantitative evaluation against advanced persistent threats," IEEE Transactions on Software Engineering, vol. 49, no. 3, pp. 1359-1373, 2022.

[4] A. Sharma, B. B. Gupta, A. K. Singh, and V. Saraswat, "Advanced Persistent Threats (APT): evolution, anatomy, attribution and countermeasures," Journal of Ambient Intelligence and Humanized Computing, vol. 14, no. 7, pp. 9355-9381, 2023.

[5] S. Garg, R. K. Singh, and A. K. Mohapatra, "Analysis of software vulnerability classification based on different technical parameters," Information Security Journal: A Global Perspective, vol. 28, no. 1-2, pp. 1-19, 2019.

[6] F. Lomio, E. Iannone, A. De Lucia, F. Palomba, and V. Lenarduzzi, "Just-in-time software vulnerability detection: Are we there yet?," Journal of Systems and Software, vol. 188, 111283, 2022.

[7] V. Satyanarayana, and M. Sekhar, "Static analysis tool for detecting web application vulnerabilities," Int. Journal of Modern Engineering Research (IJMER), vol. 1, no. 1, pp. 127-133, 2011.

[8] K. Goseva-Popstojanova, and A. Perhinschi, "On the capability of static code analysis to detect security vulnerabilities," Information and Software Technology, vol. 68, pp. 18-33, 2015.

[9] I. A. Shah, S. Rajper, and N. ZamanJhanjhi, "Using ML and Data-Mining Techniques in Automatic Vulnerability Software Discovery," International Journal of Advanced Trends in Computer Science and Engineering, vol. 10, no. 3, pp. 32-27, 2021.

[10] R. Zhang, S. Huang, Z. Qi, and H. Guan, "Static program analysis assisted dynamic taint tracking for software vulnerability discovery," Computers & Mathematics with Applications, vol. 63, no. 2, pp. 469-480, 2012.

[11] W. Wang, F. Dumont, N. Niu, and G. Horton, "Detecting software security vulnerabilities via requirements dependency analysis," IEEE Transactions on Software Engineering, vol. 48, no. 5, pp. 1665-1675, 2020.

[12] I. Chowdhury, and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities," Journal of Systems Architecture, vol. 57, no. 3, pp. 294-313, 2011.

[13] S. Rehman, and K. Mustafa, "Software design level vulnerability classification model," International Journal of Computer Science and Security (IJCSS), vol. 6, no. 4, 238, 2012.

[14] G. Huang, Y. Li, Q. Wang, J. Ren, Y. Cheng, and X. Zhao, "Automatic classification method for software vulnerability based on deep neural network," IEEE Access, vol. 7, pp. 28291-28298, 2019.

[15] X. Sun, L. Li, L. Bo, X. Wu, Y. Wei, and B. Li, "Automatic software vulnerability classification by extracting vulnerability triggers," Journal of Software: Evolution and Process, vol. 36, no. 2, e2508, 2024.

[16] J. Chen, P. K. Kudjo, S. Mensah, S. A. Brown, and G. Akorfu, "An automatic software vulnerability classification framework using term frequency-inverse gravity moment and feature selection," Journal of Systems and Software, vol. 167, 110616, 2020.

[17] Q. Wang, Y. Li, Y. Wang, and J. Ren, "An automatic algorithm for software vulnerability classification based on CNN and GRU," Multimedia Tools and Applications, vol. 81, no. 5, pp. 7103-7124, 2022.

[18] X. Li, L. Wang, Y. Xin, Y. Yang, Q. Tang, and Y. Chen, "Automated software vulnerability detection based on hybrid neural network," Applied Sciences, vol. 11, no. 7, 3201, 2021.

[19] H. Wu, H. Dong, Y. He, and Q. Duan, "Smart contract vulnerability detection based on hybrid attention mechanism model," Applied Sciences, vol. 13, no. 2, 770, 2023.

[20] X. Sun, L. Li, L. Bo, X. Wu, Y. Wei, and B. Li, "Automatic software vulnerability classification by extracting vulnerability triggers," Journal of Software: Evolution and Process, vol. 36, no. 2, e2508, 2024.

[21] P. Ganesh, Y. Chen, X. Lou, M. A. Khan, Y. Yang, H. Sajjad, P. Nakov, D. Chen, and M. Winslett, "Compressing large-scale transformer-based models: A case study on bert," Transactions of the Association for Computational Linguistics, vol. 9, pp. 1061-1080, 2021.

[22] J. Gao, H. Zou, F. Zhang, and T. Y. Wu, "An intelligent stage light-based actor identification and positioning system," International Journal of Information and Computer Security, vol. 18, no. 1/2, 204-218, 2022.

[23] T.-Y. Wu, C.-M. Chen, X. Sun, S. Liu, and J. C.-W. Lin, "A Countermeasure to SQL Injection Attack for Cloud Environment," Wireless Personal Communications, vol. 96, no. 4, pp. 5279-5293, 2016.

[24] H. Ma, T.-Y. Wu, M. Chen, R.-H. Yang, and J.-S. Pan, "A Parse Tree-Based NoSQL Injection Attacks Detection Mechanism," Journal of Information Hiding and Multimedia Signal Processing, vol.8, no.4, pp.916-928, 2017.

[25] X. Zhang, X. Dai, L. Liu, and T. Feng, "Chinese short text classification model with multi-head self-attention mechanism," Journal of Computer Applications, vol. 40, no. 12, 3485, 2020.

[26] Y. Wang, Y. Zhang, Y. Zhang, L. Zhao, X. Sun, and Z. Guo, "SARD: Towards scale-aware rotated object detection in aerial imagery," IEEE Access, vol. 7, pp. 173855-173865, 2019.

[27] D. R. Edwards, M. M. Handsley, and C. J. Pennington, "The ADAM metalloproteinases," Molecular Aspects of Medicine, vol. 29, no. 5, pp. 258-289, 2008.

[28] N. Segnan, S. Minozzi, A. Ponti, C. Bellisario, S. Balduzzi, M. González-Lorenzo, S. Gianola, and P. Armaroli, "Estimate of false-positive breast cancer diagnoses from accuracy studies: a systematic review," Journal of Clinical Pathology, vol. 70, no. 4, pp. 282-294, 2017.

[29] M. Schubert, M. Mashkour, C. Gaunitz, A. Fages, A. Seguin-Orlando, S. Sheikhi, A. H. Alfarhan, S. A. Alquraishi, K. A. Al-Rasheid, and R. Chuang, "Zonkey: A simple, accurate and sensitive pipeline to genetically identify equine F1-hybrids in archaeological assemblages," Journal of Archaeological Science, vol. 78, pp. 147-157, 2017.