The Journal of Systems & Software 198 (2023) 111591

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A mutual embedded self-attention network model for code search ☆

Haize Hu *, Jianxun Liu *, Xiangping Zhang, Ben Cao, Siqiang Cheng, Teng Long

School of Computer Science and Technology, Hunan University of Science and Technology, Hunan 411100, China
Hunan Provincial Key Lab. for Services Computing and Novel Software Technology (Hunan University of Science and Technology), Hunan 411100, China

ARTICLE INFO

Article history:
Received 20 December 2021
Received in revised form 2 December 2022
Accepted 18 December 2022
Available online 7 January 2023

Keywords:
Code search
Code segments
Machine learning
Self-attention
MESN-CS

ABSTRACT

To improve the efficiency of program implementation, developers can selectively reuse the previously written code by searching the open-source codebase. To date, many code search methods have been proposed to actively push the limit of code search accuracy, where the methods designed using Self-Attention mechanism are particularly promising. However, while existing methods can improve the efficiency to capture textual semantics by attending significant words in the code component unit, they typically fail to capture the structural dependencies between the code components which may produce suboptimal search accuracy. In this paper, we propose a novel Self-Attention model termed MESN-CS which considers both word-level attention and code unit-level attention for code search. MESN-CS not only the attention weight of each word in the code component unit is calculated, but also the weight of the embedding between the code combination units is calculated. To verify the effectiveness of the proposed model, three benchmark models were compared on a large-scale code data and CodesearchNet. The experimental results show that the MESN-CS has better *Recall@k*, *NDCG* and *MRR* performance than baseline methods. the experiments also show that the semantic syntactic information between sequences can be effectively characterized in MESN-CS.

1. Introduction

With the development of software engineering, the capacity of open-source code repository (such as GitHub) is expanding, which lays the foundation for code reuse (Kim et al., 2018). To improve the efficiency of software development, developers spend a lot of time searching existing similar code in the open-source codebase (Kim et al., 2010). The accuracy of code search is directly related to the effectiveness of code reuse (Liu et al., 2020a). Therefore, how to design a high accuracy code search model has been the focus of code search (Sivaraman et al., 2019).

Code search initially focused on the Information Retrieval (IR)-based methods, where the source code can be viewed as natural language to be matched using key words. However, the IR-based approach does not consider the semantic gap between natural language and source code, which may result in suboptimal search results. Gu et al. (2018) introduced deep learning into code search for the first time and proposed a DeepCS code search model, which uses a long–short memory network (LSTM) to embed natural language and source code into the same vector space. Gu et al. achieved an accuracy of 47.5% and initiated a new research direction for code search. To compensate for the relationship between two words in the source code ignored in DeepCS, Fang et al. (2021) proposed to introduce the self-attentive model into the code search research for the first time and proposed the SAN-CS code search model. SAN-CS achieved an accuracy of 93.1%, which is a great breakthrough for code search research relative to Gu et al. However, SAN-CS ignores the logical relationship between source code sequences, which has important feature information in the source code context. In the SAN-CS model, the source code is parsed as Method name sequence, API sequence, Token sequence and Description sequence. Take the following source code (Fang et al., 2021) as an example.

```
/**
##open a file and output the contens
*/
Public void readFile(String path) {
  File file = new File(path);
  FileReader fr = new FileReader(file)
  BufferedReader fr = new FileReader(file)
  String line = " ";
  While (null !=(line = br.readLine())){
  System.out.println(line);
  }
}
```

☆ Editor: Aldeida Aleti.

* Correspondence to: North Second Ring Road, Yuhu District, Xiangtan City, Hunan Province, China.
E-mail addresses: 952259775@qq.com (H. Hu), ljx529@gmail.com (J. Liu).

https://doi.org/10.1016/j.jss.2022.111591
0164-1212/

The Method names in the source code are *readFile*(*String path*), *File(path)* and *FileReader(file)*, the APIs are *br.readLine() and System.out.println(line)*, Description is "*open a file and output the contents*". All the keywords in the whole source code constitute Token.

The source code is parsed according to the SAN-CS model to obtain four sequences divided as follows:

Description sequence: *open, file, output, contens.*

Method name sequence: *read, File.*

API sequence: *br, read, line, print.*

Token sequence: *open, file, output, contens, read, File, path, br, read, line, print, string, null, system.*

Where, Description illustrates of the source code function "open a file and output the contens", Method name represents the entire source code function. "readFile" Description corresponds to the content of Method name. Method name ("readFile") represents the whole source code, covering "*br.readLine(), System.out.println(line).* Description as "*open a file and output the contens*". Token contains *File(path), FileReader(file)*" and keywords in the source code. For "tokens", the composition consists of Description, Method name, and API together, Therefore, there is a certain structural correlation between the four sequences. However, SAN-CS does not consider the relationship between these four sequences, and simply splices the four sequences, ignoring the correlation between the sequences.

To compensate for the deficiencies of the SAN-CS, a mutual embedded Self-Attention network code search model (MESN-CS) is proposed in the paper, and the data set (Gu et al., 2018) crawled from the open-source codebase is used for the experimental analysis. On the basis of the SAN-CS, the analysis of the embedding among the three parts of API, Token and Method name was added. The mutual embedded (Gu et al., 2018) research was used to make up for the internal weight (Fang et al., 2021), which was ignored in the SAN-CS.

To evaluate the effectiveness of the MESN-CS for code search, four models of DeepCS (Gu et al., 2018), SAN-CS (Fang et al., 2021), CSDA (Ren et al., 2020) and CARLCS-CNN (Shuai et al., 2020) were used as benchmark models for comparative analysis. In addition, five evaluation indicators, *MRR, NDCG, R@1, R@5* and *R@10*, were used to evaluate the models.

In summary, the main contributions of this paper are as follows:

a. A mutually embedded Self-Attention network code search model (MESN-CS) was proposed in the paper, which can establish semantic correlation mapping between code fragments and query language through inter-embedding Self-Attention.

b. An experimental analysis was carried out on the data set, and the effectiveness of the five models in code search were compared and analyzed. The experimental results show that the proposed model in the paper is more effective in the mapping between code fragments and query language.

The remaining structure of the article is as follows: In the second section, definition of the problems is discussed. In the third section, the preliminary involved in deep learning is discussed, which mainly includes embedding technology and trained neural network model; In the fourth section, the MESN-CS model and training method mentioned in the paper were introduced; In the fifth section, the experiments of each model were introduced and the experimental results were presented; In the sixth section, the discuss of the MESN-CS model mentioned in the paper were discussed; Lastly, the work of the full text was summarized.

2. Definition of the problems

2.1. Code search definition

Code search facilitates various code development tasks such as code writing, code refactoring and bug fixing, which can largely improve the efficiency of code development. There are two main types of functionality that developers face in the process of software development. One is the development of existing features and the other is the development of new features. For the development of existing features, instead of spending a lot of time to redevelop them, they can be queried on open source code repositories or open source communities. By modifying and reusing the query results, the development time of existing code can be reduced and more time can be spent on the development of new functional code. The process of querying the open source repository or community is referred to code search, as shown in Fig. 1. In the search tool, the query language used to describe the function of the code is entered, which in turn outputs the corresponding matching code. Then, the developer selects the output for reuse based on the results.

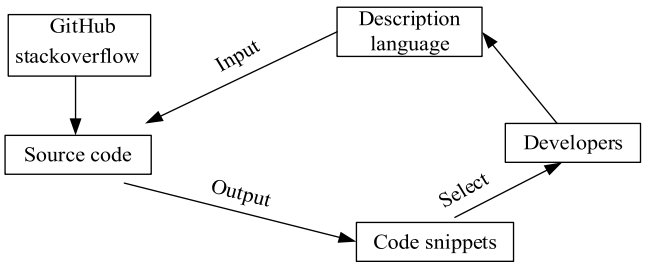

Fig. 1. Framework for code search.

2.2. Related concepts

Query statement. To perform a query on existing code, the developer enters the description language of the desired code in the search tool, which is a query statement.

Code comment. During the process of software writing, developers annotate the corresponding part of the code to facilitate subsequent checking and learning by others. In the study of code search, the offline training part focuses on matching training with code description and code fragments by replacing the query language with code comment.

Training model. The most important step in the research of code search is to match the programming language with natural language. The model is being trained given different code fragments, which in turn provides the search model for subsequent code search.

Code vector. There is a large semantic gap between programming language and natural language, and both parts need to be embedded into the same vector space by the neural network model. Therefore, the code vector is the code vector obtained from the source code fragment by model embedding.

Code preprocessing. In the research of code search, if the whole code fragment is directly embedded, it will lead to many features cannot be extracted effectively. Therefore, the source code fragments need to be preprocessed first. At present, the main research preprocessing methods include abstract syntax tree parsing (Shuai et al., 2020), graph structure processing (Kim et al., 2010), serialization (Gu et al., 2018), etc.

2.3. Motivation

Gu et al. firstly parsed source code into API Sequence, Method-name Sequence, and Token Sequence, and introduced deep learning into code search task for the first time, and proposed a DeepCS code search model. Sequence parsing methods have been widely used by researchers, and the latest research is the SAN-CS model proposed by Fang et al. SAN-CS makes up for the deficiency of

DeepCS in source code syntax structure and largely improves the accuracy of code search. However, the relationship between the three partial sequences of API, methodname, and token were ignored in SAN-CS. There are certain logical structure information between API sequences, Methodname sequences, and Token sequences, which was very important for the characterization of code snippets.

To characterize the structural information between the above three sequences, we propose a MESN-CS code search model by extracting the structural information between sequences using the mutual embedding technique. First, we analyze the correlation between API Sequence, Methodname Sequence, and Token Sequence, and discuss the impact of the structural information between different sequences on the code search task. Second, we train and test the model on the DeepCS model experimental dataset and the CodesearchNet (which is widely used for code search tasks) dataset to verify the effectiveness of MESN-CS in code search tasks. Finally, we analyze the parameter settings of the model and discuss the model search speed in comparison. The experimental results show that There are some structural correlations among API Sequence, Methodname Sequence, and Token Sequence, and the structural information of different sequences has different effects on the results, among which the structural information of Method name and Token has the greatest effect on MESN-CS. Therefore, MESN-CS can accurately compensate for the structural information between sequences, improving the accuracy of source code characterization and thus facilitating the development of code search.

3. Background knowledge

3.1. Code embedding

The programming language and query language cannot be recognized by the neural network directly for training. Before being trained, the programming language and query language need to be converted into vectors through an embedding technology. Therefore, embedding technology is a bridge between language input and neural network.

The initial word embedding was mainly based on the one-hot coding model (Shi et al., 2019), which is a high-dimensional and high-sparse coding method. The high dimensionality is because the total number of words was used as the coding dimension, and the high sparsity is because each code has only one bit of feature information (1 represents the feature position, and the others were 0) (Yuan et al., 2020). Due to the characteristics of one-hot encoding, the larger the amount of text data, the higher the sparsity of its encoding (Yin and Shen, 2018; Feng et al., 2020b). Based on the high sparsity of one-hot encoding, word embedding was proposed (Chen et al., 2016; Yang et al., 2015). Word embedding maps a word or phrase to a n-dimensional numerical vector. Because of its advantages, word embedding technology has been widely used by most researchers. For example, word embedding models were used in bilingual testing (Xing et al., 2015) and cross-language research (Ruder et al., 2019). The most representative word embedding model is word2vec (Church, 2017), which converts an n-dimensional one-hot vector into an m-dimensional vector of spatial real numbers, thus reducing the vector dimension. The word2vec model has two structures: Continuous Bags-of-Words (CBOW) and Skip-Gram (SG) (Ling et al., 2015). To verify the effectiveness of the proposed model, the embedding model-CBOW, which was used in Gu et al. (2018), Fang et al. (2021) and Ren et al. (2020), is used in the paper.

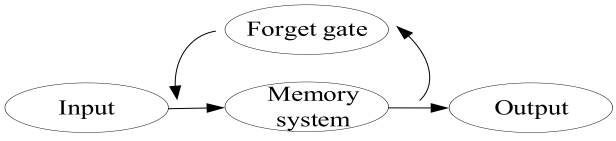

Fig. 2. Structure of LSTM.

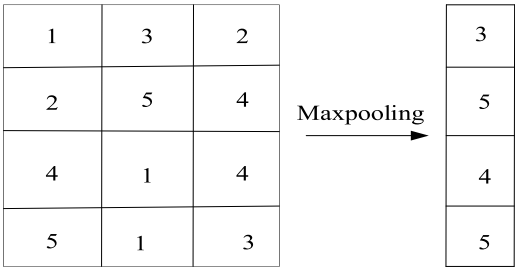

Fig. 3. Examples of maximum pooling.

3.2. Training model

In this part, the widely used neural network LSTM (Greff et al., 2016; Zhao et al., 2017) will be introduced. LSTM is a special recurrent neural network (RNN) that can selectively store important information and can effectively compensate for the long-term dependence of RNN. The structure of LSTM is shown in Fig. 2:

The input gate determines whether information is input to the memory cell, the output gate determines whether the memory cell outputs information, and the forget gate determines the information that can be forgotten in the memory cell. The mathematical principle of LSTM is as follows.

$$\begin{aligned} f_t &= \sigma(W_{xf}x_t + W_{hf}x_{h-1} + b_f) \\ i_t &= \sigma(W_{xi}x_t + W_{hi}x_{h-1} + b_i) \\ o_t &= \sigma(W_{xo}x_t + W_{ho}x_{h-1} + bo) \\ \bar{c}_t &= \tanh(W_{xc}x_t + W_{hc}x_{h-1} + b_c) \\ c_t &= f_t \otimes c_{t-1} + i_t \otimes \bar{c}_t \\ h_t &= o_t \otimes \tanh(c_t) \end{aligned} \tag{1}$$

Where, W_{xf}, W_{hf}, W_{xi}, W_{hi}, W_{xo}, W_{ho}, W_{xc}, W_{hc} represent the corresponding weight vector between the input vector, input gate, memory unit, and output gate respectively, b_f, b_i, b_o, b_c are bias variables, $\otimes$ is the matrix product.

LSTM is a time series training network. The vector value of the last word is used as the result of all information training, and the feature is extracted by maximum pooling. The maximum pooling calculation is as follows.

$$s = \max pooling(h_1, h_2, \ldots, h_t) \tag{2}$$

Where, h is the output of the hidden layer of the training model, and t is the number of training individuals. The maximum pooling is shown in Fig. 3.

The cosine similarity was used to calculate the correlation, and the cosine similarity was calculated as follows.

$$\cos(V_{code}, V_{query}) = \frac{V_{code} \bullet V_{query}}{\|V_{code}\| \bullet \|V_{query}\|} \tag{3}$$

To train the correlation between the code segment and the description, it is necessary to prepare a pair of descriptions for each code segment during the training process, including a positively related description and a negatively related description. The word embedding and neural network training are performed on both at the same time. The purpose of training is to make

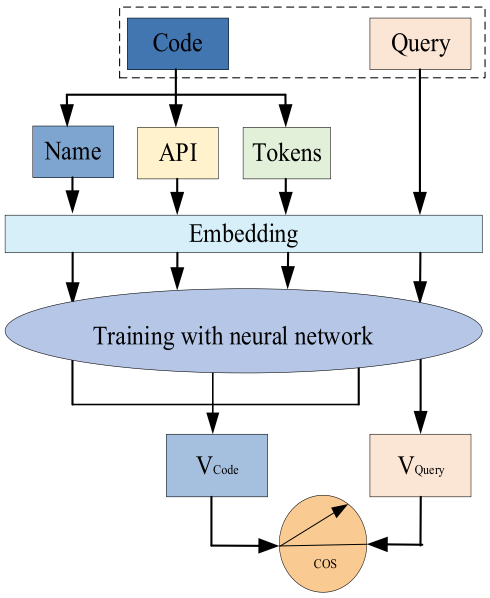

Fig. 4. Embedding process of code search.

the code have the larger correlation with the positive correlation description as possible, while the negative correlation description has the smaller correlation. The ranking loss was used to calculate the objective function as follows.

$$L(\theta) = \max(0, \xi - \cos(V_{code}, V^{+}_{query}) + \cos(V_{code}, V^{-}_{query})) \tag{4}$$

Where, θ is the training model parameter, V_{code} is the code vector, V^{+}_{query} is the positive correlation description vector, V^{-}_{query} is the negative correlation description vector, and ξ is the training model parameter. The code snippets embedding and query sentence embedding are shown in Fig. 4.

3.3. Self-attention model

Self-Attention is a kind of attention mechanism, which was mainly used in text processing fields such as machine translation. In the training process, since the weight of each input is considered, the Self-Attention can extract more accurate features. The execution process of the Self-Attention is as follows.

(1) Suppose a text sequence S_n= $[m_1, m_2, m_3, \ldots, m_n]$, which is composed of n words, and embed these n words into a word vector v_n= embedding(m_n).

(2) The three weight matrices W_{key}, W_{value}, W_{query} are randomly initialized, and each word vector v_n is multiplied to obtain three own vectors K_n, V_n, Q_n, which are called key vector, value vector, and query vector respectively. The calculation process is shown below.

$$\begin{aligned} K_n &= m_n \bullet W^{T}_{key} \\ V_n &= m_n \bullet W^{T}_{value} \\ Q_n &= m_n \bullet W^{T}_{query} \end{aligned} \tag{5}$$

(3) The Q_n and the K_n are multiplied and divided by the vector dimension to obtain *atten*. Each word vector performs this step to get its own *atten*, as follows.

$$atten = \frac{Q_n \bullet K^{T}_{n}}{\sqrt{d}} \tag{6}$$

(4) The *atten* calculates in step 3 was subjected to *softmax* calculation, and then multiplied by V_n to obtain the influence value score of each word vector on the current processing vector, as follows.

$$score = soft\max(atten) \bullet V_n \tag{7}$$

(5) Take the first word vector v_1 as an example, and assume that the influence of the n word vector on v_1 is $score_1$. The v_1 can be obtained by adding each score, and the output vector value (v_{out}) is obtained after processing by the Self-Attention mechanism.

$$v_{out} = \sum_{i=1}^{n} score_i \tag{8}$$

(6) After repeating steps 3, 4, and 5, the output vector of each word vector processed by the Self-Attention mechanism will be obtained.

4. MESN-CS code search model

The code search research framework is mainly divided into two parts, which are the offline training part and the online search part. The offline part of the research is to get the corresponding matching model, which is being trained by a large amount of code data, and the online part is to rely on the model obtained from the offline part as a search tool for online search. The paper is based on the offline training part of the research and analysis, through which an efficient code search model is obtained. The model proposed in the paper is shown in Fig. 5.

The "Dataset" in Fig. 5 is obtained by crawling techniques on open source code repositories or open source communities. Moreover, to simulate the matching training between the training query statements and the programming language, the crawled code must contain code description. Therefore, code and comments exist in pairs in the code fragment dataset. To improve the accuracy of source code and code annotation feature extraction, the source code is pre-processed before the model is trained. This source code preprocessing is done in the form of type serialization (Method name sequences, API sequences, Tokens sequences and Description sequences). The source code fragments are preprocessed by serialization to obtain API sequences, Method name sequences, Tokens sequences and Des sequences (code description sequences), and the four sequences obtained by preprocessing are sent to MESN-CS for training. In the MESN-CS model, the sequences are embedded and vectorized by the deep neural network, and then the API sequence vector V_{API}, Method name sequence vector V_{name}, Tokens sequence vector V_{Tokens}, and code description sequence vector V_{Des} are obtained in the characterization stage. The 3 sequences in the code fragment were fully concatenated to obtain the code fragment vector V_{Code}, and finally vector matching was performed by cosine similarity. In the vector matching process, 2 code annotation vectors need to be set to form a matching pair with a code vector at the same time. The two code description are the positive correlation annotation vector $V_{+\text{Des}}$ and the negative correlation annotation vector $V_{-\text{Des}}$, respectively. The positively correlated annotation vector is derived from the annotation vectors of code pairs in the dataset, and the negatively correlated vector is randomly derived from the annotation vectors of codes of other code pairs. The goal of the model training is to make the code vector V_{Code} match well with the positive correlation vector $V_{+\text{Des}}$, while match poorly with the negative correlation vector $V_{-\text{Des}}$. The optimal parameters of the MESN-CS model were calculated by setting the loss degree function during the training process.

After the training, the optimal MESN-CS model parameters are obtained. To evaluate the effectiveness of the model, the dataset used is distinguished from the training test, which is not used

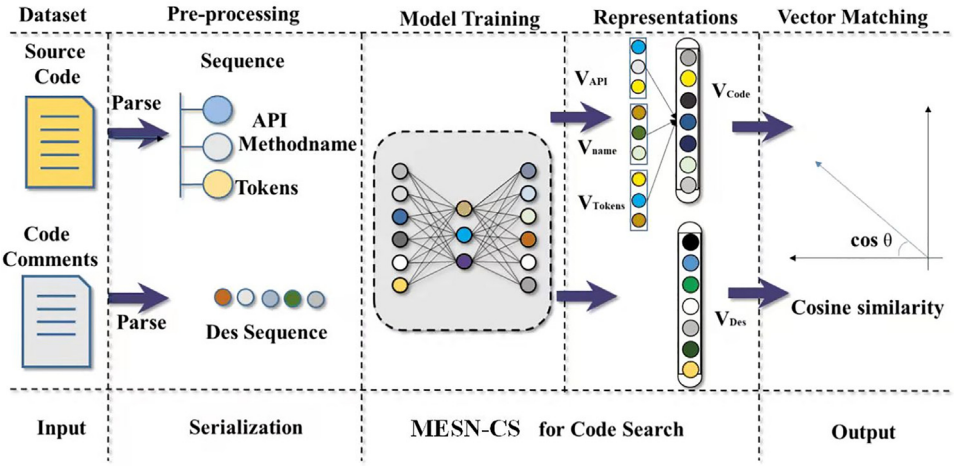

Fig. 5. Framework of MESN-CS.

in the training part. Similarly, the code and code description are defined as test pairs, and MESN-CS is used for feature extraction and vectorization of the source code and code description, respectively. Finally, the matching calculation is performed using cosine similarity to test the validity of the model.

4.1. MESN-CS model

Fig. 6 is the MESN-CS training model of the code search proposed in the paper. There are two parts of input in the training model. The first part of the input is the code snippet, and the second part of the input is the description in the code snippet. The code snippet consists of three parts, API, Method name (hereinafter abbreviated as name), and Token. The training steps are as follows:

The first step is embedding. The sequences of API, Name, Token and Description are embedded to obtain the vectors VAPI, Vname, Vtoken and VDes. Through this step, the programming language and natural language are embedded in the same vector space.

The second step is Self-Attention training. Through the four partial vectors obtained by embedding in the first step, the internal feature weight of the sequence was considered.

The third step is mutual-embedding training. For the code snippet of AAPI, AName, AToken, the mutual-embedding technology is used to obtain MAAPI, MAName, AToken (Fig. 6 only lists the API and Name mutual-embedding, and the other two groups have the same principles as API and Token, Name and Token).

The fourth step is full connection. The mutual-embedding vectors obtained in the third part are spliced to obtain the vector Vcode of the entire code fragment.

The fifth step is similarity calculation. Mutual embedding attention is used to select the weight of the code fragment vector Vcode and the description vector ADes to obtain MVcode and MADes. Finally, average pooling is used to extract the features, and the cosine similarity is used to calculate the similarity.

The main goal of the proposed MESN-CS model is to address the sequence correlation ignored by the SAN-CS model, and the mutual embedding technique is employed to compensate for the sequence correlation. Through the mutual embedding model, the correlation between sequences is added in the model to improve the overall code representativeness. Meanwhile, to further improve the code representation information and reduce the loss of feature information, the residual network was employed in the model training process. The initial values of the feature information vector are superimposed on the feature information extraction vector by the residual network to target the information lost in the training process. In MESN-CS, not only the correlation between sequences is taken into account, but also the loss of training features is compensated, which in turn improves the accuracy of the whole code feature extraction.

On the basis of word embedding, a sequence embedding that integrates multiple words together is proposed (Bespalov et al., 2012). Sequence embedding technology is mainly used to characterize and train code fragments and query sentences for code search. The code snippet needs to be pre-processed before the code snippet was embedded (Gu et al., 2018; Fang et al., 2021; Ren et al., 2020), and the entire code snippet was split into three parts: Method name, API and Token (as show in Fig. 7). We extracted the Method name and used the camel case to resolve the Method name into a sequence. After the Method name sequence and API sequence were segmented, they are added to the Token sequence.

After the sequences were embedded, the sequence vectors are obtained. And the vectors of Method name sequence and API sequence are fed into the LSTM neural network as input for feature extraction. Due to the Token sequence does not reflect the contextual logic of the source code, the multi-layer perceptron (MLP) (Riedmiller and Lernen, 2014) is used to perform vector feature extraction. Finally, the three-part feature vector obtained through the fully connected layer to obtain the final code embedding vector. Query (description) sequence embedding was similar to code embedding, with the difference that the code language is replaced with natural language. The query sequence is embedded to obtain the query sequence vector, and LSTM is used for vector feature extraction.

Gu et al. performed Method name, API, Token and Description (des) extraction on the source code, and the sequence extraction is performed as follows (Gu et al., 2018).

For Method name sequence. The name of each source code fragment is extracted and the Method name was cut into a sequence of Tokens by camel case.

For API sequence: The Eclipse JDT compiler (Kim et al., 2010) is used to parse the code, which is generated according to the following rules (Gu et al., 2018).

(a) For a particular constructor *new. C()*, *C.new* is generated and added to the API sequence. For a method *o.m()*, the *C.m* sequence is generated and added to the API sequence.

(b) For a method call passed as a parameter, the method type is prefixed to that parameter method, e.g. $o_1.m_1$. It produces an append sequence $C_2.m_2 - C_3.m_3 - C_1.m_1$, where C_i is the class of the instance o_i.

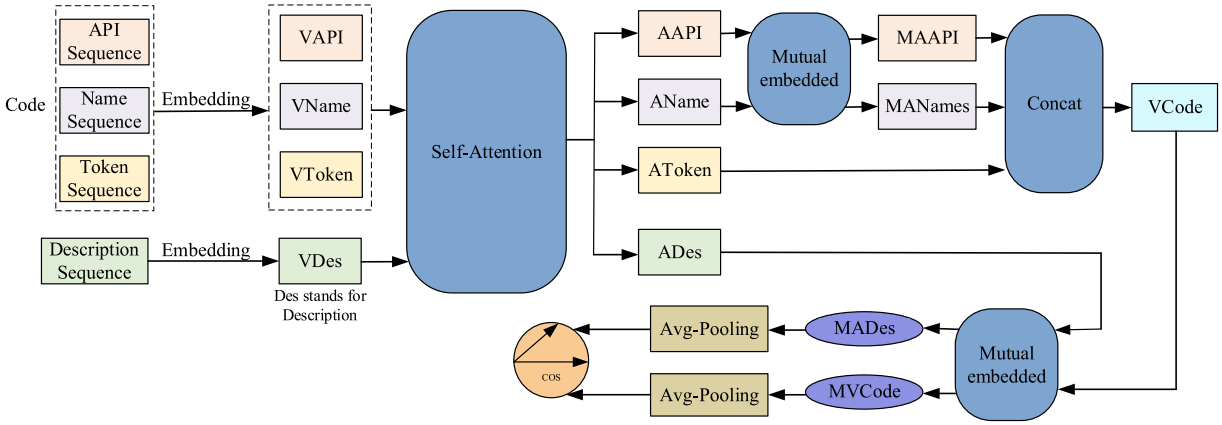

Fig. 6. MESN-CS for code search.

```
static void copy(String scr, String dest)throws IOException{
            while ((readBytes = fis.read(b))>0) fos.write(b, 0, readBytes);
            fis.close();
            fos.close();
        }
#Methodname
#API
#token
```

Fig. 7. Examples of code structures.

Table 1
Strengths and weaknesses of the models.

Model	Can be characterized	Cannot be characterized/shortcomings
DeepCS	Programming language and natural language are embedded in the same space.	The correlation between "void" and "close" in a token sequence.
SAN-CS	Based on the DeepCS, the correlation between "void" and "close" in a token sequence.	The correlation between "copy" and"fis.read" in Method name and API sequence, respectively.
MESN-CS	Based on the SAN-CS, the correlation between "copy" and"fis.read" in Method name and API sequence, respectively.	The model has huge parameters and relies on large data

(c) For the series of statements, $a_1, a_2, \ldots, a_N$ sequences are generated. For conditional statements, if a_1 then a_2, otherwise a_3, then the API sequence is generated as a_1-a_2-a_3.

For Token sequence: Camel case is used to parse the entire source code and remove deactivated words, repetitive words and generic words. As shown in Fig. 8, the source code is parsed into individual words, and generic words (words with less semantic information) are removed, e.g., private, static, return, etc.

For Des sequence: The Eclipse JDT compiler is used to parse the source code AST and extract the source code comments from the AST. To better present the results of the four sequence extractions, an example is shown in Fig. 8.

To further demonstrate the effectiveness of the proposed model in the paper, the code fragment in Fig. 8 is presented as an example. Due to the large amount of all the information presented, only some examples of models are shown in Table 1.

4.2. Code embedding

According to the DeepCS, the source code fragment is split into four sequences: API sequence, Method name sequence, Token sequence and Description sequence. Therefore, each sequence needs to be embedded separately, and four sequence vectors V_{API}, V_{name}, V_{token} and V_{Des} are obtained through embedding technology. For the sake of simplicity, Method name is abbreviated as (n), API is abbreviated as (a), Token is abbreviated as (t), and Description was abbreviated as (D).

4.2.1. Self-attention training

There are 4 sequence embeddings in the process of training. The first is the embedding of the Method name sequence, assuming that the original code name sequence is $s = \{s_1, s_2, \ldots, s_n\}$, where n is the length of the sequence. So, there are n name words in the sequence. The previous Section 3.3 has explained the basic principles of the Self-Attention model. The Method name sequence is trained to obtain the query vector Q_n, the key vector K_n, and the value vector V_n through the Self-Attention model, as follows.

$$\begin{aligned} Q_n &= S_n \bullet W_Q^T \\ K_n &= S_n \bullet W_K^T \\ V_n &= S_n \bullet W_V^T \end{aligned} \tag{9}$$

The Self-Attention weight *att()* of each component of the Method name sequence is calculated as follows, and *Softmax()* is used to calculate the weight value:

$$\begin{aligned} context_n &= Att(Q_n, K_n) \bullet V_n \\ Att(Q_n, K_n) &= SoftMax(\tfrac{Q_n \bullet K_n^T}{\sqrt{d}}) \end{aligned} \tag{10}$$

Where, d is the dimension, and *SoftMax()* is the weight parameter matrix.

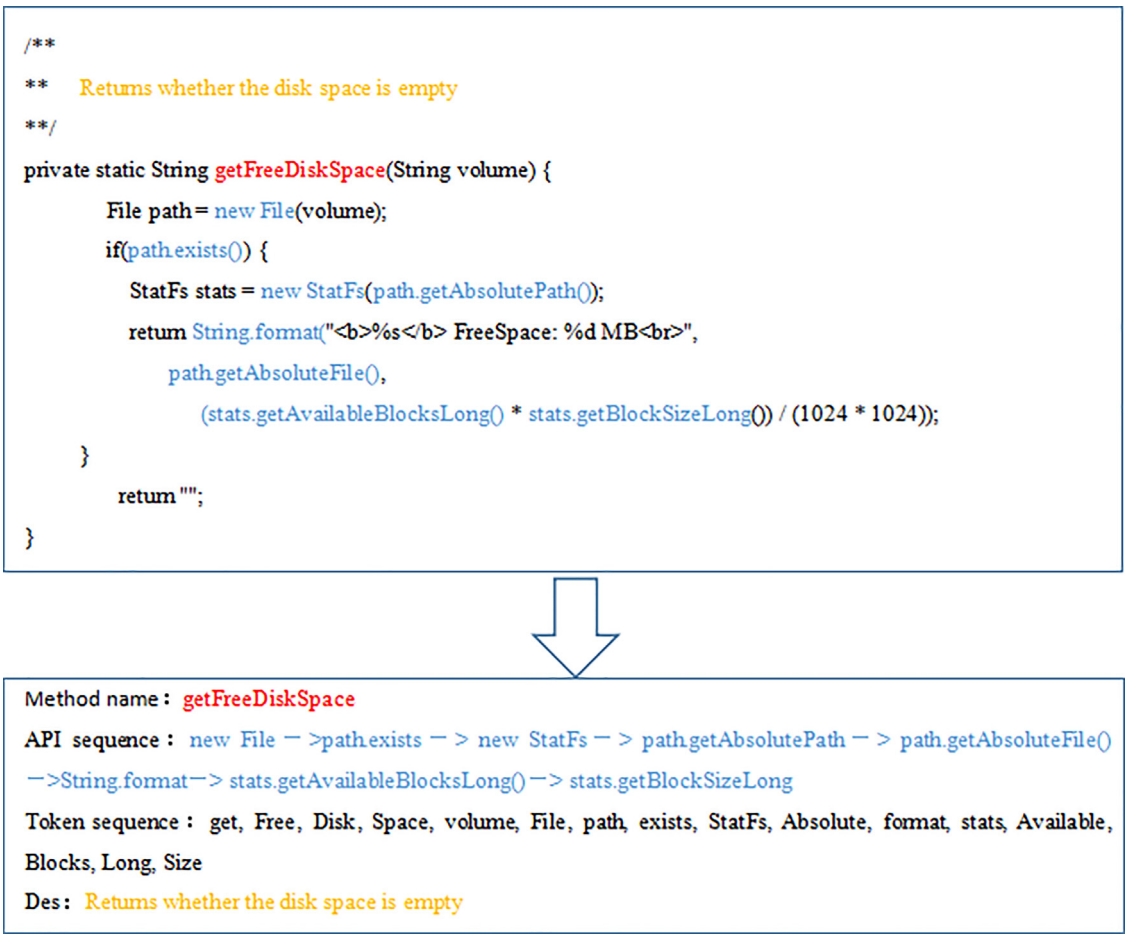

Fig. 8. Results of source code sequence extraction.

The *ReLu* activation function is used to further extract the feature information, and through the fully connected layer calculation, the Method name sequence vector is obtained, as follows:

$$v_{name} = \mathrm{ReLu}(context_n \bullet w_1 + b_1) + b_2 \tag{11}$$

Where, *ReLu()* is the activation function, W_1 and W_2 are the weight parameters, and b_1 and b_2 are the weight parameters.

In the same way, we can obtain the API sequence vector V_a, Tokens sequence vector V_t, Description sequence vector V_D, respectively.

$$v_{\mathrm{a}} = \mathrm{ReLu}(context_a \bullet w_1 + b_1) + b_2 \tag{12}$$

$$v_{\mathrm{t}} = \mathrm{ReLu}(context_t \bullet w_1 + b_1) + b_2 \tag{13}$$

$$v_{\mathrm{D}} = \mathrm{ReLu}(context_t \bullet w_1 + b_1) + b_2 \tag{14}$$

4.2.2. Mutual-embedding research

Deep learning was first applied to code search research for the first time, and LSTM network neural network was used for training (Gu et al., 2018). In view of the shortcomings of LSTM training, a code search model with a Self-Attention model SAN-CS as a training network was proposed (Fang et al., 2021). However, the SAN-CS only considered the attention calculation between the internal words of each sequence of API, Method name, Token and Description. The paper ignores that the code snippet was a whole, and there is a certain logical structure correlation among the four sequences. Aiming at the deficiencies in SAN-CS, a Self-Attention model based on inter-embedding is proposed, which is called the MESN-CS model in this paper. The mutual-embedding model is based on the Self-Attention model and considered the embedding between two sequences each time. Therefore, the MESN-CS considered three types of mutual-embedding. Mutual-embedding of Method name and API, mutual-embedding of Method name and Token, mutual-embedding of API and Token.

Mutual-embedding of Method name and API.

$$\begin{aligned} Q_n &= V_n \bullet W_{na}^T \\ V_n &= V_n \bullet W_{nn}^T \\ k_a &= V_a \bullet W_{ka}^T \\ V_a &= V_a \bullet W_{na}^T \end{aligned} \tag{15}$$

The parameter a of the mutual-embedding matrix:

$$A = SoftMax(\frac{Q_{\mathrm{n}} \bullet k_d^T}{\sqrt{d}}) \tag{16}$$

The API embedded Method name is V_n, and the Method name embedded API is V_a, which is obtained as follows:

$$\begin{aligned} V_n &= A \bullet V_a \\ V_a &= A \bullet V_n \end{aligned} \tag{17}$$

Method name and Token mutual-embedding:

$$\begin{aligned} Q_n &= V_n \bullet W_{nt}^T \\ V_n &= V_n \bullet W_{nn}^T \\ k_t &= V_t \bullet W_{kt}^T \\ V_t &= V_t \bullet W_{nt}^T \end{aligned} \tag{18}$$

The parameter a of the mutual-embedding matrix:

$$A = SoftMax(\frac{Q_t \bullet k_d^T}{\sqrt{d}}) \tag{19}$$

The Token embedded Method name is V_n, and the Method name embedded Token is V_t, which is obtained as follows:

$$\begin{aligned} V_n &= A \bullet V_t \\ V_t &= A \bullet V_n \end{aligned} \tag{20}$$

API and Token mutual-embedding:

$$\begin{aligned} Q_a &= V_a \bullet W_{ta}^T \\ V_a &= V_a \bullet W_{aa}^T \\ k_t &= V_t \bullet W_{kt}^T \\ V_t &= V_t \bullet W_{at}^T \end{aligned} \tag{21}$$

The parameter a of the mutual-embedding matrix:

$$A = SoftMax(\frac{Q_a \bullet k_d^T}{\sqrt{d}}) \tag{22}$$

The Token embedded API is V_a, and the API embedded Token is V_t, which is obtained as follows:

$$\begin{aligned} V_a &= A \bullet V_t \\ V_t &= A \bullet V_a \end{aligned} \tag{23}$$

Through the splicing function, the code fragment vector V_c is obtained as follows:

$$V_c = concat(V_n, V_a, V_t) \tag{24}$$

The Self-Attention model considers the weight of each word in the sequence, so maximum pooling is not suitable. Because average pooling can effectively represent all internal information, average pooling is used in the process of pooling (Song et al., 2018; Shen et al., 2018). The average pooling of code and Description are as follows:

$$\begin{aligned} V_{cav} &= avgpooling([V_{c1}, V_{c1}, \ldots, V_{cn}]) \\ V_{Dav} &= avgpooling([V_{D1}, V_{D1}, \ldots, V_{Dn}]) \end{aligned} \tag{25}$$

Where, the average value of code vector is V_{cav}, and the average value of code description vector is V_{Dav}.

4.3. Model training

Code comments are used as descriptions during offline training, and query statements are used as descriptions during online search, so that the target of training is converted from the match between the query sentence and the code to the match between the code and the code description. To train the matching between the code and the description, a positive correlation code description vector V_{Dav}^+ and a negative correlation code description vector V_{Dav}^- are set (Gu et al., 2018; Fang et al., 2021; Ren et al., 2020). The goal of training is to make the code vector have a high similarity with the positive correlation code description vector V_{Dav}^+ and a low similarity with the negative correlation code description vector V_{Dav}^-. The minimum ranking loss function was used to construct the objective function (van der Laan and Gruber, 2016; Saha et al., 2021), which was calculated as follows:

$$L(\theta) = \sum_{(c,d^+,d^-)} \max(0, \xi - \cos(c, d^+) + \cos(c, d^-)) \tag{26}$$

Where, θ is the model parameter, $cos(c, d^+)$ and $cos(c, d^-)$ are the cosine similarity between the code fragment and the positive and negative correlation description, and ξ is the boundary constant. In the process of loss function training, the Adam optimizer (Zhang, 2018; Mehta et al., 2019) optimizer is used for iterative optimization.

Table 2
Data sets 1.

Dateset	Size	Language
Train	18233872	Java
Valid	100000	Java
Test	10000	Java

Table 3
Data sets 2.

Dateset	Size	Language
Train	454451	Java
Valid	15328	Java
Test	26909	Java
All	496688	Java

5. Experimental evaluation

5.1. Experiment preparation

In this part, the preparations for the experimental evaluation are mainly described, including the data set used in the experimental evaluation, the introduction of the characterization parameters of the experimental evaluation, and the introduction of the comparison model.

All the comparison experiments were carried out on a server with two Nvidia GTX 2080Ti GPUs. The number of GRU units in the proposed method is 128, dropout rate is 0.2.

5.1.1. Experimental data set

To enhance the comparative reliability of the experiments, the dataset crawled by Gu et al. (2018) (Table 2) was used. Gu et al. crawled all source code snippets for the Java language from August 2008 to June 2016 on the open source platform GitHub. To ensure the quality of the source code snippets, the source code without stars were removed. For the task of experimentation, the source codes were crawled with code comments. Summing up the requirements of the above dataset, a total of 18,233,872 source code fragments were obtained.

Meanwhile, to further validate the effectiveness of the proposed MESN-CS, we used the Java dataset in CodesearchNet for experimental analysis (Training and testing of the model). The composition of the data is shown in Table 3. The total Java dataset in CodesearchNet in Table 3 is 496,688 entries, of which the training set is 454,451, the validation set is 15,328, and the test set is 26909.

5.1.2. Evaluating parameters

The purpose of the test experiment is to validate the code search model and evaluate the search effectiveness of the model. The code search task is essentially a simulation of a developer entering a query statement into the search model, and the search model recommends the corresponding code snippet from the code base based on similarity calculation. Among the recommended code snippets, the one with the highest similarity result will be ranked first in the output results, and the one with lower similarity will be ranked last. Finally, developers select the code they need from the output results according to their needs. And, the higher the correct source code is in the output, the higher the score of the result. Thus, the code search task is essentially a search ranking problem, and the correct source code is in the output results, representing that the search model is valid.

To evaluate the code search model MESN-CS proposed in the paper, code search related indicators are used for evaluation. At the same time, combined with the SAN-CS, three parameter indicators are used in the paper for evaluation, which are recall

rate (*Recall*) (Nafi et al., 2019; Li et al., 2017), standardized return (*NDCG*) (Niu et al., 2017; Husain et al., 2019), average reciprocal (*MRR*) (Yunianto et al., 2020; Kahil, 2020). In the evaluation index of recall rate, *Recall*@1, *Recall*@5, and *Recall*@10 were selected as the evaluation of recall rate.

Recall is used to detect the index position corresponding to the optimal result in the query output result. The earlier the index result, the more effective the model efficiency (Yan et al., 2018). *Recall@k* is used as the metric value, where the values of k are 1, 5, and 10 respectively. The calculation is as follows:

$$\text{Recall@k} = \frac{1}{|Q|}\sum_{J=1}^{|Q|}\varepsilon \tag{27}$$

Where, Q is the data sequence of the test data set, and ε is the query parameter. When the query result k value contains the optimal result, the ε result takes the value 1, otherwise it is defined as 0.

NDCG is measured on the basis of *recall*, which can characterize the relevance of search code fragments and query sentences (Drain et al., 2021). The calculation is as follows:

$$NDCG = \frac{1}{|Q|}\sum_{j=1}^{k}\frac{2^{r_j}-1}{\log 2(1+j)} \tag{28}$$

Where, r_j is the relevance of the j position in the recall rate. A larger *NDCG* indicates higher quality search results, as well as higher rankings.

MRR represents the score of the first occurrence of the desired result value among the results (Wen et al., 2017). The calculation is as follows:

$$MRR = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{Index_{Qj}} \tag{29}$$

Where, the score of $Index_{Qj}()$ represents that the first output is the best and the jth word in the query Q_j.

5.1.3. Benchmark model

To analyze the effect of the proposed model MESN-CS in the task of code search, the models proposed in Gu et al. (2018), Fang et al. (2021), Ren et al. (2020) and Shuai et al. (2020) are used for comparative analysis. The DeepCS model represents the first study of introducing deep learning into code search, and provides a new research direction for code search research. The CARLCS-CNN is proposed on the basis of DeepCS, and the CNN network is used to replace the LSTM network to further improve the accuracy of code search. The SAN-CS is proposed based on the DeepCS. The Self-Attention network is proposed for code search research to improve the shortcomings of LSTM from the perspective of word attention. Therefore, DeepCS, CARLCS-CNN,CSDA and SAN-CS are used as benchmark models for comparative analysis. The baseline models are as follows.

DeepCS: Gu et al. (2018) introduced deep learning into code search research for the first time and used LSTM to embed both code language and natural language into a vector.

SAN-CS: Based on DeepCS, Fang et al. (2021) used Self-Attention instead of LSTM to further consider the logical relationship between the data.

CSDA: Based on DeepCS, Ren et al. (2020) used an attention network to optimize Tokens sequences, further considering the weights between data.

CARLCS-CNN: Based on the DeepCS study, Shuai et al. (2020) used a CNN network to replace the LSTM and a common attention network to construct the mutual matrix, which considers the dependencies between the code and the query statements.

Table 4
Experimental results.

Model	*Recall*@1	*Recall*@5	*Recall*@10	*MRR*	*NDCG*
DeepCS	0.4752	0.7610	0.8633	0.6169	0.6169
CARLCS-CNN	0.5491	0.7132	0.7824	0.5354	0.5926
CSDA	0.6574	0.7241	0.7854	0.7012	0.7315
SAN-CS$^-$	0.5953	0.7133	0.8143	0.5773	0.6331
SAN-CS	0.9310	0.9560	0.9620	0.9080	0.9210
MESN-CS	**0.9410**	**0.9642**	**0.9716**	**0.9207**	**0.9329**

Table 5
Results of mutual embedding impact.

Model	*Recall*@1	*Recall*@5	*Recall*@10	*MRR*	*NDCG*
SAN-CS	0.9310	0.9560	0.9620	0.9080	0.9210
MESN-NA	0.9412	0.9644	**0.9742**	0.9209	0.9331
MESN-NT	**0.9419**	**0.9656**	0.9720	**0.9214**	**0.9338**
MESN-AT	0.9399	0.9627	0.9687	0.9197	0.9317

5.2. Experimental results

5.2.1. Comparative experiment

The experiment is mainly based on the data set (Section 5.1.1), with a reference model (Section 5.1.3) as an experimental object, and characterizes the model effect with the evaluation index parameters (Section 5.1.2).

The code snippet consists of three parts: API, Method name, and Token, there are three mutual-embedding models: API and Method name mutual-embedding (MESN-NA), API and Token mutual-embedding (MESN-AT), Method name and Token mutual-embedding (MESN-NT). To compare the effects of MESN-CS, DeepCS (Gu et al., 2018), CSDA (Ren et al., 2020), CARLCS-CNN (Shuai et al., 2020), SAN-CS- and SAN-CS (Fang et al., 2021) are used for comparison.

The experiment consists of three main steps. Firstly, the model parameters are trained based on a large dataset, and the intrinsic parameters of the model were obtained after training. Secondly, the model is validated based on the trained model using a cross-experimental approach. To ensure the model is being properly trained, 10-fold cross validation was employed to validate the model and further fine-tune the model parameters. Finally, the model obtained from the training is tested and the model effect is tested using a small dataset. During the experimental analysis, the training dataset, validation dataset and test dataset are obtained separately without crossover data between them.

The experimental results of MESN-CS in Table 4 are the average values under the three mutual embeddings, and each experiment result is based on the average of five experiments. Compared with DeepCS, CARLCS-CNN, CSDA, SAN-CS-, and SAN-CS models, MESN-CS shows better results. For *Recall@1*, the improvements are 98.02%, 71.37%, 43.14%, 58.07%, and 1.07%, respectively; for *Recall@5*, the improvements are 26.70%, 35.19%, 33.16%, 35.17%, 0.86%, respectively; for *Recall@10*, the improvements are 12.54%, 24.18%, 24.18%, 19.32%, 1.00%, respectively; for *MRR*, the improvements are 49.25%, 71.96%, 31.30%, 59.48%, respectively 1.40%; for *NDCG*, the improvements are 51.22%, 57.42%, 27.53%, 47.35%, and 1.29%, respectively.

To further analyze the impact of mutual embedding on the model, ablation experiments are used for analysis, the test results under different embeddings are analyzed separately. The experimental results are shown in Table 5.

As can be seen in Table 5, the model has the best results under the Method name and Token inter-embedding. The experimental results show that the interrelationship between Method name and Token contains the most important information in the process of code characterization. Compared with the experimental results of SAN-CS, the relationship between Method name and API

Table 6
Mutual embedding of API and Name.

k	*Recall@k*	*MRR*	*NDCG*	*Query time*
1	0.9412	**0.8862**	**0.8862**	0.0458
5	0.9644	0.9199	0.9307	0.0741
10	0.9704	0.9209	0.9331	0.1026

Table 7
Mutual embedding of API and Token.

k	*Recall@k*	*MRR*	*NDCG*	*Query time*
1	0.9399	0.8853	0.8853	0.0290
5	0.9627	0.9188	0.9295	0.0625
10	0.9687	0.9197	0.9317	0.0911

Table 8
Mutual embedding of Name and Token.

k	*Recall@k*	*MRR*	*NDCG*	*Query time*
1	**0.9419**	0.8859	0.8859	**0.0289**
5	**0.9656**	**0.9205**	**0.9315**	**0.0578**
10	**0.9720**	**0.9214**	**0.9338**	**0.0868**

Table 9
Parameter of MESN-CS.

Parameter	*batch_size*	*n_heads*	*d_word_dim*	*n_layers*	*dropout*	*learning_rate*
Value	128	8	128	2	0.2	$1e^{-5}$

Table 10
Result of batch_size.

batch_size	*Recall@1*	*Recall@5*	*Recall@10*	*MRR*	*NDCG*
64	0.9332	0.9572	0.9612	0.9187	0.9278
128	**0.9410**	**0.9642**	**0.9716**	**0.9207**	**0.9329**
256	0.9401	0.9614	0.9698	0.9198	0.9302

and the relationship between API and Token contain rich semantic and syntactic information. However, the information between sequences is ignored in SAN-CS, so the proposed model in this paper effectively improves the accuracy of code representation.

To further illustrate the impact of different mutual embedding on the model, the experimental results of three groups of mutual embedding are compared and analyzed respectively. The experimental results are shown in Tables 6–8. It can be seen from the results that the model results under the mutual embedding of token and names are more accurate and the query time is lower.

5.2.2. Parameter adjustment

In the course of the experiments, the corresponding parameter adjustments are made to obtain the optimal model parameters. The optimal experimental results obtained in Section 5.2.1 are based on the optimal parameters, and the main parameters of the MESN-CS model are specified in Table 9.

In Table 9, *batch_size* is the number of data trained in each batch of the model, *d_word_dim* is the dimensionality of the word embedding, *n_heads* is the number of multi-headed self-attentive feature types, *n_layers* is the number of neural network layers of the model, and *learning_rate* and *dropout* are the learning rate and loss rate of the model, respectively.

For the parameter *batch_size*, we have compared three values of 64, 128, and 256, respectively. 64, 128, 256, and 512 are more frequently used in comparative research applications, but due to the limitations of our experimental environment (insufficient memory), we were unable to experiment with *batch_size* = 512. The results of the experiment are shown in Table 10.

For the parameter *d_word_dim*, the values of 64, 128 and 256 are currently studied and applied, so we only analyze the three values in the comparison process. Since the multi-headed attention model is used, for *d_word_dim* = 64, the *n_heads* = 8, which is defined as 1*type; for d_word_dim = 64, the *n_heads* = 16, which is defined as 2*type; for *d_word_dim* = 128, the *n_heads* = 8, which is defined as 3*type; for *d_ word_dim* = 128, at this time *n_heads* = 16, defined as 4* type; the results of the experiment are shown in Table 11.

Table 11
Result of d_word_dim.

d_word_dim	Recall@1	Recall@5	Recall@10	MRR	NDCG
1*	0.9032	0.9124	0.9356	0.8965	0.8996
2*	0.9278	0.9365	0.9512	0.9012	0.9091
3*	**0.9410**	**0.9642**	**0.9716**	**0.9207**	**0.9329**
4*	0.9369	0.9547	0.9681	0.9201	0.9314

Table 12
Result of n_layers.

n_layers	Recall@1	Recall@5	Recall@10	MRR	NDCG
1	0.8786	0.8963	0.8741	0.8205	0.8746
2	**0.9410**	**0.9642**	**0.9716**	**0.9207**	**0.9329**
3	0.9112	0.9223	0.9354	0.9036	0.9014

Table 13
Result of dropout.

dropout	Recall@1	Recall@5	Recall@10	MRR	NDCG
0.1	0.9012	0.9147	0.9258	0.9012	0.9147
0.2	**0.9410**	**0.9642**	**0.9716**	**0.9207**	**0.9329**
0.3	0.9349	0.9564	0.9614	0.9187	0.9289

Table 14
Comparison of effects.

Model	Parameters	Training time	Text time
DeepCS	5.85M	1.00 ms/sample	0.60 s/query
SAN-CS	6.64M	0.30 ms/sample	0.10 s/query
MESN-CS	7.47M	0.57 ms/sample	0.07 s/query

For the parameter *n_layers*, we have compared and analyzed three commonly used values, 1, 2, and 3. The results of the experiment are shown in Table 12.

For the parameter *dropout*, we have compared and analyzed three commonly used values, which are 0.1, 0.2, and 0.3. The results of the experiment are shown in Table 13.

5.2.3. Efficiency analysis

To further analyze the efficiency of MESN-CS, the number of model parameters, training time and query time are compared and analyzed separately. DeepCS introduces deep learning to code search research for the first time, and SAN-CS introduces self-attentive models to code search for the first time. DeepCS and SAN-CS have made a big step forward for code search research and contributed greatly to the research of code search. Therefore, DeepCS and SAN-CS are mainly used as comparative analysis models in this section. The analysis results of the experiments are shown in Table 14.

In Table 14, "Parameters" indicates the total number of parameters that can be trained in the model during training. The size of the parameters represents the size of the memory required for model training and determines the hardware conditions required for the model. The "training time" and "text time" are the time required to train and test the model once, respectively, and the size of the time represents the speed of training and testing of this model and determines its efficiency.

As can be seen from Table 14, MESN-CS has 1.62 M and 0.83 M more parameters than DeepCS, MESN-CS, respectively. During the training process, the training time is 43.0% faster than DeepCS, MESN-CS, and 90.0% slower than SAN-CS. During testing, MESN-CS has the fastest test time, 88.33% and 30.0% faster than DeepCS

Table 15
Experimental results.

Model	*Recall*@1	*Recall*@5	*Recall*@10	*MRR*	*NDCG*
DeepCS	0.2432	0.2654	0.3123	0.2414	0.2784
SAN-CS	0.9085	0.9273	0.9478	0.8630	0.8748
MESN-CS	0.9239	0.9411	0.9593	0.8812	0.8906

and SAN-CS, respectively. Therefore, MESN-CS has a higher search effect.

5.2.4. Adaptability analysis

To further explore the effectiveness of the MESN-CS model in the task of code search, we further experimentally analyze on the dataset in Table 3. Then, to ensure the reliability of the experimental results, we use the average of five experiments as the final data results. Compared with the dataset in Table 2, the number of datasets in Table 3 is small, so we adjust the *learning_rate* value to 5e-5 and the *dropout* value to 0.1 for the parameters of the model, while keeping the other parameters unchanged. The experimental results are shown in Table 15.

As can be seen from the table, the model is slightly less effective on Table 3 compared to Table 2. However, DeepCS performs very poorly based on Table 3, which also indicates that the DeepCS model may need to rely on the training of large datasets to ensure the accuracy of the search. For Recall@1, Recall@5, Recall@10, MRR and NDCG, MESN-CS improves 1.7%, 1.5%, 1.2%, 2.1% and 1.8% over SAN-CS, respectively. In summary, the experimental results not only show that the MESN-CS model performs better on the Java dataset of CodesearchNet, butit also shows that the adaptability of the model is based on an amount of data training.

6. Discussion

6.1. Why MESN-CS model is more effective

We have identified 2 advantages of MESN-CS, which can explain its effectiveness in code search.

a. Better semantic understanding of utterances by Self-Attention. Unlike traditional deep learning models, MESN-CS uses a self-attentive model for joint embedding of source code and natural language utterances. The self-attentive model is able to focus not only on word meaning, but also to better characterize the contextual logical relationships between words.

b. Better source code integrity through sequence mutual embedding. For the first time, MESN-CS takes into account the semantic correlation between sequence utterances, which has been neglected in historical studies. Since Chen et al. (2016) first proposed to parse source code in three sequences, API, Token and Method name, subsequent researchers have not considered the semantic correlation between the three sequences and thus ignored the source code integrity. MESN-CS analyzes the semantic correlation between the three sequences and verifies the importance of source code integrity through experiments.

6.2. Threats to validity

In this section, we discuss threats to the validity of their study in general, external validity and construct validity, and discuss how we mitigate these threats.

a For model internal validity, it is mainly the threat of overlap between training and testing data that makes the accuracy of the model unconvincing. To reduce the threat of internal validity, we use three independent data for model training, validation and testing. And to ensure the reliability of the dataset, we select the most widely searched projects in Github and remove the projects that do not contain stars.

b For the threat of model external validity, this refers to whether the proposed MESN-CS can be generalized to other scenarios of application. In the experimental analysis, we mainly based on the Java dataset crawled by Gu et al. The data used for the experiments are created individually by Gu et al. There is a great deal of personal habit and therefore the data set is not universal in composition. Therefore, there are certain threats and challenges in the validity and generality of the model. To reduce the threat of external validity of the model, we select the Java dataset in CodesearchNet, a common dataset for code search, for analysis and validate the effectiveness of MESN-CS in code search research. At present, MESN-CS can only process the Java dataset in CodesearchNet, and we will continue to explore the generalization of MESN-CS to other languages such as Python, C, and go in future studies.

c For the threat of model construct validity, it mainly refers to whether the structure of the source code can reflect the real application scenario. Because the dataset used in our experimental process exists in code-comments pairs, then whether the components can truly describe the code is a threat factor that affects the validity of MESN-CS. To reduce the threat of construct validity of MESN-CS, we leverage several Java programmers to manually check the 10,000 test datasets and modify them for comments inaccuracies to improve the accuracy of the tests. In future studies, we will manually check all datasets, which is a time-consuming process.

6.3. Implication

The article is a study of code search and provides 2 implications.

a. A certain research direction has been assigned to code search from the perspective of model optimization.

At present, the research on code search mainly focuses on the exploration of training models, and various optimization models have been proposed. However, the results of existing models for code search are similar, and the improvement of accuracy has encountered certain bottlenecks. Therefore, the paper chooses to build on the existing models to study and analyze them for further accuracy improvement. To improve the accuracy of the existing models and break the bottleneck problem of the accuracy of the existing models. As the number of proposed training models continues to increase, the continuous optimization of existing models will become a new direction for future research. Thus, the paper provides a new direction for code search research staff.

b. Providing model reference for software development research.

The key problem of code search research is the study of the semantic gap between programming language and natural language, and the research focuses on how to accurately match information between programming language and natural language through vectors. The research in the paper is not only applicable to programming language, but also adapted to the processing of natural language. Therefore, although the paper is only based on code search, the deep learning neural network model is equally applicable to research based on matching between code and natural language and matching between code and code. Among them, the code-to-natural language processing matching research contains studies on automatic code annotation extraction, code nature prediction, code defect detection, and code style improvement. The code-to-code matching study includes code clone detection, code completion, automatic program repair, etc.

7. Related work

For the research of code search, the design of the model has been the focus of attention, and the accuracy of the code search results is improved through the design of the model. For the initial code search research, information retrieval technology was widely used for matching research (Hersh, 2021). For example, the syntax tree and information retrieval technology were combined for similarity detection to improve the accuracy of code search (Karnalim, 2020). Information retrieval technology was used to enhance machine learning for similarity detection, and then to code clone detection research (Hammad et al., 2020). Different information retrieval models and software engineering tasks were matched and studied (Rahman et al., 2019). Information retrieval technology based on semantic information was studied for fuzzy matching of codes (Jain et al., 2021). Information retrieval technology was used for code similarity detection (Karnalim et al., 2019), which was used in the research of code plagiarism. However, information retrieval technology uses simple semantic information to characterize the code, does not consider the difference between the programming language and natural language, and also ignores the structural information in the source code (C and Shenoy, 2020). Therefore, in the task of improving the accuracy of code search, the effect of information retrieval technology is still limited (Sachdev et al., 2018).

Due to certain shortcoming and deficiency in information retrieval technology, Gu et al. (2018) introduced deep learning into code search research for the first time. Long short-term memory (LSTM) network was used to extract features of code information and code comments, and the source code and code comments were trained through neural network for similarity matching. Through deep learning, programming language and natural language were mapped to a common vector space to solve the semantic gap between programming language and natural language (Cambronero et al., 2019).

Code search research was opened up in a new direction with the research of Gu et al. Following the research, a large number of code search methods based on deep learning models appeared. Wen et al. (2020) proposed a multi-level semantic representation method MSR for maximum semantic matching from the representation of words and text. Sun et al. (2022b) proposed a semantic and syntactic filter to improve the accuracy of the search model, which uses deep learning to effectively refine and extract the semantics and syntax of source code. Ren et al. (2020) proposed a CSDA training model in order to distinguish the intrinsic characteristics of different sequences, and CNN networks and attention networks were employed for feature extraction of token sequences to improve the accuracy of token sequence feature extraction. Wu and Yan (2022) used a combination of attention, aggregation vectors and intermediate representations to improve the effectiveness of code search from a multimodal perspective, and validated it in CodeSearchNet (Ren et al., 2020). Yu et al. (2022) proposed a deep code search model SQ-DeepCS based on code structure and quality. Gotmare et al. (2021) proposed a classifier cascade model, Cascade, to improve the accuracy of final code search by a two-level classifier. Source code being processed into text or abstract syntax tree (AST) are the two main methods of preprocessing, but both methods have certain shortcomings in the characterization of code semantics. Zeng et al. (2021) proposed a deGraphCS model for code search by converting source code into an intermediate variable flow graph and training it with an LSTM neural network. However, control flow graphs and abstract syntax trees tend to lead to different embeddings of the same word when multiple word mapping functions are utilized. Sun et al. (2022a) proposed a context-aware translation model, TRANSS, which translates source code fragments into natural language using a vocabulary, and then performed search matching. Shi et al. (2021) proposed a multi-modal control contract code search model MM-SCS to effectively capture the data flow and control flow in the code graph, which improved the effectiveness of code semantic feature extraction by focusing more on important contexts. Liu et al. (2021) analyzed that graph neural networks need to rely on the global nature of the graph in the process of construction, which made graph neural networks have great limitations in the process of learning features. Further, a GraphSearchNet joint learning model was proposed to optimize the global dependency of BiGGNN graphs through a multi-headed attention mechanism to improve the feature learning ability of the model. Li et al. (2022) proposed two code feature extraction models. On one hand, a method based on the text and API was proposed for feature extraction. On the other hand, code comments and codes were employed to train the model for pairwise feature extraction. Due to the single nature of the model parameters, it cannot show superior results on multiple data sets. Ling et al. (2020) proposed an AdaCS model to solve the problem of high cost of code research, which can effectively transfer training between different code databases. Feng et al. (2020a) used pre-training for feature extraction of source code and proposed the pre-training model CodeBERT, and applied it to code search research.

However, the LSTM model was used to process information by Gu, which cannot effectively characterize code features. When dealing with longer texts, LSTM is unable to correlate long distance information and is prone to information loss. At the same time, the source code was split into three parts (Gu et al., 2018): Method name, API and Token, and the weight analysis of the internal information of each part was ignored. To make up for the shortcomings of the proposed model by Gu, various optimization models have been proposed. For example, Shuai et al. (2020) considered that the content of Method name sequence and Query language sequence is short, and LSTM cannot effectively extract information. Therefore, convolutional neural network (CNN) is used to replace part of LSTM, which effectively improves the accuracy of code search. However, only a part of the practicability of LSTM has been solved, and the weight of internal feature information has not been analyzed and considered. Based on the proposed model by Fang et al. (2019) considered the relationship between the word information in the token sequence. In the process of training, the attention layer was increased, and the weight of each token in the sequence was calculated. However, the internal weight between API and Method name was ignored in the model. In the proposed model by Liu et al. (2020b), Method name, Token and keywords were used for matching, which effectively improved the speed of code search. A hierarchical attention network and convolutional neural network combined optimization model was proposed (Wang et al., 2020). However, the proposed model did not consider the weight of each component in the code source. In addition, multi-layer attention models have been added to improve the effectiveness of code search. Inspired by those models, a Self-Attention network model (SAN-CS) was proposed by Fang et al. (2021). The Self-Attention network was used to replace the LSTM network for training, and the internal feature weight was fully considered (Shaw et al., 2018). In the SAN-CS, not only the internal weights of the three parts were considered, but also the mutual embedding weights between the programming language and the natural language were considered. However, the connection between the three parts of Method name, API and Token was ignored.

In summary, the research on code search has evolved from the initial introduction of information retrieval to the continuous application of deep learning. The characteristics of several major code search models are shown in Table 16, where information retrieval is abbreviated as IR, machine learning as ML, and deep learning as DL.

Table 16
Differences in code search models.

Model		Advantage	Shortcoming
IR	AST+IR	Compared to traditional information retrieval, the structural information of the source code is taken into account.	The structural information is too homogeneous to highlight the important internal feature information. Meanwhile, the semantic gap between source code and natural language cannot be considered, resulting in poor model search.
	ML+IR	The accuracy of the search results is improved by machine learning algorithms.	Over-reliance on mathematical models, the choice of mathematical model has a large impact on the search results. Moreover, it does not have the ability to learn the feature information of the source code, and the optimal result can only be obtained by successive iterative calculations.
	TF-IDF	Local and global relationships are considered, important feature information is highlighted.	Heavy reliance on the source code corpus. If the word list of the corpus is not rich enough, it leads to the inaccurate weighting of certain words. Moreover, the same word in different corpus will lead to incompatible weights due to the inconsistency of its word types. All these will lead to inaccurate word weights in the source code representation process.
DL	DeepCS	For the first time, deep learning is introduced to code search, embedding natural language and source code into the same vector space to largely improve the semantic divide problem.	Contextual structural information in the source code is not taken into account and the word-to-word logic in the source code is not well thought out. Also, the weight of words in the source code is not considered.
	CSDA	Semantic features are distinguished	Only the weights within the token sequences are considered, however, no consideration is given to the API and Method name sequences. Also, the logic between sequences is ignored and the integrity of the source code is lacked to be considered.
	CARLCS-CNN	Sequence are considered separately	The weights between sequences are ignored, lacking consideration of source code integrity. Also, the relationship between words within sequences is not considered.
	SAN-CS	Introducing Self-Attention to the code search for the first time and getting consideration for the internal word-to-word logic of the sequence.	Only the word-to-word logic within the sequence is considered, while the relationships between the sequences are ignored, which will result in the integrity of the source code not being guaranteed.
	DeGraphCS	Better Semantic Syntax Representation	The model parameters are too large and occupy a large amount of memory, while the training time of the model is too long. The graph-based representation lacks the correlation representation between graphs and diagrams.
	CodeBERT	Enables training of small data to overcome the dependence on big data training.	The pre-training data and fine-tuning data are highly variable, and the characteristics of the fine-tuning data directly affect the accuracy of the test results. Meanwhile, the fine-tuning data are generally small, which makes it difficult to learn the task accurately.

8. Conclusion

In the paper, a new code search model, referred to as the MESN-CS model, was proposed based on the mutual embedding Self-Attention mechanism. Firstly, the self-attentive network mechanism was utilized in the early model to find the self-attentive weights of the code sequences, and then the information of the sequences' own features was obtained. Then the mutual embedding model was used to derive the weights between code sequences, and thus the semantic correlation between code sequences was extracted. The comparative experimental results showed that the MESN-CS model has better performance than benchmark models in the evaluation indexes *Recall@k*, *MRR*, and *NDCG*. Therefore, the MESN-CS can be effectively adapted to the study of code search. The results of the ablation experiments show that there is rich representation information between the sequences (Method name, API and Token). The proposed MESN-CS can effectively extract the feature information between sequences in the code. MESN-CS effectively improves the accuracy of code characterization and the effectiveness of code search.

The next step of the research is planned.

a. Simplification of the model. Due to the huge model parameters in training, it needs to occupy a large capacity space. In the next research, we will try to simplify the model. On the basis of ensuring the accuracy of the results, the model structure will be simplified, the model parameters will be reduced, and thus the experimental cost will be reduced.

b. Exploration of training models for small data sets. Based on migration learning techniques, we will explore small dataset training models to improve experimental efficiency.

c. Seeking industrial collaboration. In real industry, we will look for partners with code search engineering applications to validate the validity of the model in real engineering scenarios.

CRediT authorship contribution statement

Haize Hu: Conceptualization, Methodology, Software, Writing – original draft, Data curation. **Jianxun Liu:** Writing – review & editing, Funding acquisition. **Xiangping Zhang:** Supervision, Validation. **Ben Cao:** Project administration, Visualization. **Siqiang Cheng:** Data curation, Formal analysis. **Teng Long:** Formal analysis, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported by the Natural Science Foundation of China, under grants 61866013, and the Education Department Key Foundation of Hunan Province in China, under grants 17A173, and the Education Department Foundation of Hunan Province in China, under grants 18C0565.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jss.2022.111591.

References

Bespalov, D., Qi, Y., Bai, B., et al., 2012. Sentiment classification with supervised sequence embedding. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, Berlin, Heidelberg, pp. 159–174.

C, H.S., Shenoy, M.K., 2020. Advanced text documents information retrieval system for search services. Cogent Eng. 7 (1), 1856467.

Cambronero, J., Li, H., Kim, S., et al., 2019. When deep learning met code search. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 964–974.

Chen, C., Gao, S., Xing, Z., 2016. Mining analogical libraries in Q & A discussions–incorporating relational and categorical knowledge into word embedding. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering. SANER, vol. 1, IEEE, pp. 338–348.

Church, K.W., 2017. Word2Vec. Nat. Lang. Eng. 23 (1), 155–162.

Drain, D., Hu, C., Wu, C., et al., 2021. Generating code with the help of retrieved template functions and stack overflow answers. arXiv preprint arXiv:2104.05310.

Fang, S., Tan, Y.S., Zhang, T., et al., 2021. Self-attention networks for code search. In: Information and Software Technology. Vol. 134. 106542.

Fang, W., Wen, X.Z., Xu, J., et al., 2019. CSDA: a novel cluster-based secure data aggregation scheme for WSNs. Cluster Comput. 22 (3), 5233–5244.

Feng, Z., Guo, D., Tang, D., et al., 2020a. Codebert: A pre-trained model for programming and natural languages. arXiv preprint arXiv:2002.08155.

Feng, C., Wang, T., Yu, Y., et al., 2020b. Sia-RAE: A siamese network based on recursive AutoEncoder for effective clone detection. In: 2020 27th Asia-Pacific Software Engineering Conference. APSEC, IEEE, pp. 238–246.

Gotmare, A.D., Li, J., Joty, S., et al., 2021. Cascaded fast and slow models for efficient semantic code search. arXiv preprint arXiv:2110.07811.

Greff, K., Srivastava, R.K., Koutník, J., et al., 2016. LSTM: A search space odyssey. IEEE Trans. Neural Netw. Learn. Syst. 28 (10), 2222–2232.

Gu, X., Zhang, H., Kim, S., 2018. Deep code search. In: 2018 IEEE/ACM 40th International Conference on Software Engineering. ICSE, IEEE, pp. 933–944.

Hammad, M., Ö, Babur., Basit, H.A., 2020. Augmenting machine learning with information retrieval to recommend real cloned code methods for code completion. arXiv preprint arXiv:2010.00964.

Hersh, W., 2021. Information retrieval. In: Biomedical Informatics. Springer, Cham, pp. 755–794.

Husain, H., Wu, H.H., Gazit, T., et al., 2019. Codesearchnet challenge: evaluating the state of semantic code search. arXiv preprint arXiv:1909.09436.

Jain, S., Seeja, K.R., Jindal, R., 2021. A fuzzy ontology framework in information retrieval using semantic query expansion. Int. J. Inf. Manage. Data Insights 1 (1), 100009.

Kahil, A.S.T., 2020. Functionality analysis and information retrieval in electronic document management systems.

Karnalim, O., 2020. Syntax trees and information retrieval to improve code similarity detection. In: Proceedings of the Twenty-Second Australasian Computing Education Conference. pp. 48–55.

Karnalim, O., Budi, S., Toba, H., et al., 2019. Source code plagiarism detection in academia with information retrieval: Dataset and the observation. Inform. Educ. 18 (2), 321–344.

Kim, K., Kim, D., Bissyandé, T.F., et al., 2018. FaCoY: a code-to-code search engine. In: Proceedings of the 40th International Conference on Software Engineering. pp. 946–957.

Kim, J., Lee, S., Hwang, S., et al., 2010. Towards an intelligent code search engine. In: Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 24. No. 1.

Li, L., Feng, H., Zhuang, W., et al., 2017. Cclearner: A deep learning-based clone detection approach. In: 2017 IEEE International Conference on Software Maintenance and Evolution. ICSME, IEEE, pp. 249–260.

Li, X., Gong, Y., Shen, Y., et al., 2022. CodeRetriever: unimodal and bimodal contrastive learning. arXiv preprint arXiv:2201.10866.

Ling, W., Dyer, C., Black, A.W., et al., 2015. Two/too simple adaptations of word2vec for syntax problems. In: Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 1299–1304.

Ling, C., Lin, Z., Zou, Y., et al., 2020. Adaptive deep code search. In: Proceedings of the 28th International Conference on Program Comprehension. pp. 48–59.

Liu, C., Xia, X., Lo, D., et al., 2020a. Opportunities and challenges in code search tools. arXiv preprint arXiv:2011.02297.

Liu, C., Xia, X., Lo, D., et al., 2020b. Simplifying deep-learning-based model for code search. arXiv preprint arXiv:2005.14373.

Liu, S., Xie, X., Ma, L., et al., 2021. GraphSearchNet: Enhancing GNNs via capturing global dependency for semantic code search. arXiv preprint arXiv:2111.02671.

Mehta, S., Paunwala, C., Vaidya, B., 2019. CNN based traffic sign classification using adam optimizer. In: 2019 International Conference on Intelligent Computing and Control Systems. ICCS, IEEE, pp. 1293–1298.

Nafi, K.W., Kar, T.S., Roy, B., et al., 2019. Clcdsa: cross language code clone detection using syntactical features and api documentation. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 1026–1037.

Niu, H., Keivanloo, I., Zou, Y., 2017. Learning to rank code examples for code search engines. Empir. Softw. Eng. 22 (1), 259–291.

Rahman, M.M., Chakraborty, S., Kaiser, G., et al., 2019. Toward optimal selection of information retrieval models for software engineering tasks. In: 2019 19th International Working Conference on Source Code Analysis and Manipulation. SCAM, IEEE, pp. 127–138.

Ren, L., Shan, S., Wang, K., et al., 2020. CSDA: A novel attention-based LSTM approach for code search. J. Phys.: Conf. Ser. IOP Publishing 1544 (1), 012056.

Riedmiller, M., Lernen, A.M., 2014. Multi layer perceptron. In: Machine Learning Lab Special Lecture, University of Freiburg, pp. 7–24.

Ruder, S., Vulić, I., Søgaard, A., 2019. A survey of cross-lingual word embedding models. J. Artificial Intelligence Res. 65, 569–631.

Sachdev, S., Li, H., Luan, S., et al., 2018. DD. In: Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages. pp. 31–41.

Saha, M., Dey, S., Wang, L., 2021. Parametric inference of the loss based index C pm for normal distribution. Qual. Reliab. Eng. Int..

Shaw, P., Uszkoreit, J., Vaswani, A., 2018. Self-attention with relative position representations. arXiv preprint arXiv:1803.02155.

Shen, W., Du, C., Jiang, Y., et al., 2018. Bag of shape features with a learned pooling function for shape recognition. Pattern Recognit. Lett. 106, 33–40.

Shi, H., Wang, R., Fu, Y., et al., 2019. Vulnerable code clone detection for operating system through correlation-induced learning. IEEE Trans. Ind. Inform. 15 (12), 6551–6559.

Shi, C., Xiang, Y., Yu, J., et al., 2021. Semantic code search for smart contracts. arXiv preprint arXiv:2111.14139.

Shuai, J., Xu, L., Liu, C., et al., 2020. Improving code search with co-attentive representation learning. In: Proceedings of the 28th International Conference on Program Comprehension. pp. 196–207.

Sivaraman, A., Zhang, T., Van den Broeck, G., et al., 2019. Active inductive logic programming for code search. In: 2019 IEEE/ACM 41st International Conference on Software Engineering. ICSE, IEEE, pp. 292–303.

Song, Z., Liu, Y., Song, R., et al., 2018. A sparsity-based stochastic pooling mechanism for deep convolutional neural networks. Neural Netw. 105, 340–345.
Sun, W., Fang, C., Chen, Y., et al., 2022a. Code search based on context-aware code translation. arXiv preprint arXiv:2202.08029.
Sun, Z., Li, L., Liu, Y., et al., 2022b. On the importance of building high-quality training datasets for neural code search. arXiv preprint arXiv:2202.06649.
van der Laan, M., Gruber, S., 2016. One-step targeted minimum loss-based estimation based on universal least favorable one-dimensional submodels. Int. J. Biostat. 12 (1), 351–378.
Wang, H., Zhang, J., Xia, Y., et al., 2020. COSEA: convolutional code search with layer-wise attention. arXiv preprint arXiv:2010.09520.
Wen, M., Chen, J., Wu, R., et al., 2017. An empirical analysis of the influence of fault space on search-based automated program repair. arXiv preprint arXiv:1707.05172.
Wen, D., Yang, L., Zhang, Y., et al., 2020. Multi-level semantic representation model for code search. In: CIRCLE.
Wu, C., Yan, M., 2022. Learning deep semantic model for code search using CodeSearchNet corpus. arXiv preprint arXiv:2201.11313.
Xing, C., Wang, D., Liu, C., et al., 2015. Normalized word embedding and orthogonal transform for bilingual word translation. In: Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 1006–1011.
Yan, X., Shi, Z., Zhong, Y., 2018. Vision-based global localization of unmanned aerial vehicles with street view images. In: 2018 37th Chinese Control Conference. CCC, IEEE, pp. 4672–4678.
Yang, H., Hu, Q., He, L., 2015. Learning topic-oriented word embedding for query classification. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, Cham, pp. 188–198.
Yin, Z., Shen, Y., 2018. On the dimensionality of word embedding. arXiv preprint arXiv:1812.04224.
Yu, H., Zhang, Y., Zhao, Y., et al., 2022. Incorporating code structure and quality in deep code search. Appl. Sci. 12 (4), 2051.
Yuan, Y., Kong, W., Hou, G., et al., 2020. From local to global semantic clone detection. In: 2019 6th International Conference on Dependable Systems and their Applications. DSA, IEEE, pp. 13–24.
Yunianto, I., Permanasari, A.E., Widyawan, W., 2020. Domain-specific contextualized embedding: A systematic literature review. In: 2020 12th International Conference on Information Technology and Electrical Engineering. ICITEE, IEEE, pp. 162–167.
Zeng, C., Yu, Y., Li, S., et al., 2021. deGraphCS: embedding variable-based flow graph for neural code search. arXiv preprint arXiv:2103.13020.
Zhang, Z., 2018. Improved adam optimizer for deep neural networks. In: 2018 IEEE/ACM 26th International Symposium on Quality of Service. IWQoS, IEEE, pp. 1–2.
Zhao, Z., Chen, W., Wu, X., et al., 2017. LSTM network: a deep learning approach for short-term traffic forecast. IET Intell. Transp. Syst. 11 (2), 68–75.

Hu Haize obtained his bachelor's degree from Hunan Institute of engineering and master's degree from Changsha University of technology in 2013 and 21016 respectively. After graduation, he has been engaged in teaching in Jishou University for four years and is now studying for a doctor's degree from Hunan University of science and technology.

Jianxun Liu, professor and doctoral advisor, obtained his bachelor's degree, master's degree and doctor's degree from Hunan Institute of engineering, Central South University and Shanghai Jiao Tong University in 1989, 1997 and 2003 respectively.

Xiangping Zhang, received master's degree and bachelor's degree from Hunan University of science and technology in 2016 and 2019 respectively. He is now studying for a doctor's degree in Hunan University of science and technology. His research interests include code representation and code clone detection.

Ben Cao, received bachelor's degree from Hunan University of Arts and Science in 2020. He is now studying for a master's degree in Hunan University of science and technology. His research interests include code representation and code search.

Siqiang Cheng, received bachelor's degree from Hunan University of Humanities in 2019. He is now studying for a master's degree in Hunan University of science and technology. His research interests include code representation and code classification.

Teng Long, received bachelor's degree from Hunan University of Science and Technology in 2020. She is now studying for a master's degree in Hunan University of science and technology. Her research interests include code representation and code completion.