The Journal of Systems and Software 207 (2024) 111856

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

Code semantic enrichment for deep code search ☆

Zhongyang Deng [a,b], Ling Xu [a,b,*], Chao Liu [a,b], Luwen Huangfu [c,d], Meng Yan [a,b]

[a] Key Laboratory of Dependable Service Computing in Cyber Physical Society (Chongqing University), Ministry of Education, China
[b] School of Big Data and Software Engineering, Chongqing University, Chongqing, China
[c] Fowler College of Business, San Diego State University, San Diego, CA, USA
[d] Center for Human Dynamics in the Mobile Age (HDMA), San Diego State University, San Diego, CA, USA

ARTICLE INFO

Dataset link: https://github.com/cqu-isse/SemEnr

Keywords:
Code search
Deep learning
Semantic enrichment
Co-attention mechanism

ABSTRACT

Code search aims to retrieve code snippets from a large-scale codebase, where the semantics of the searched code match developers' query intent. Code is a low-level implementation of programming intents, but query is always expressed as clear and high-level semantics, which makes it difficult for DL-based approaches to learn the semantic relationship between them. Through a large-scale empirical analysis on more than 2.2 million pairs of Java code and description, we found that the semantics of code and query can be aligned by enriching code with the descriptions of other code in terms of similar implementation. Based on the finding, we propose a code semantic enrichment approach for deep code search, named **SemEnr**. Specifically, we first enrich semantics for all code snippets in the training and testing data. We estimated the syntactic similarity of each code snippet from the training data and retrieved the most similar one for each. Thereafter, the semantics of one code snippet is represented by its code tokens and the description of the retrieved most similar code. During the model training, we used the attention mechanism to embed pairs of enriched code and query into the shared high-dimensional vector space. To enhance the quality of our learned representations, we integrated a multi-perspective co-attention mechanism, employing Convolutional Neural Networks (CNNs) to capture local correlations between code and query. Finally, we evaluated the effectiveness of our approach by performing experiments on two extensively used Java datasets. Our experimental results reveal that SemEnr achieves an MRR of 0.698 and 0.631, outperforming the best baseline CAT (a state-of-the-art DL-based model) by 19.93% and 18.83%, respectively. In addition, we conducted a user study involving 50 real-world queries to assess SemEnr's performance, and the findings suggest that SemEnr outperformed baseline models by returning more relevant code snippets.

1. Introduction

The large software communities, such as GitHub,[1] Stack Overflow,[2] and GitLab,[3] provide millions of publicly available source code. The substantial improvement of software development efficiency can be achieved by developers through searching online for and reusing existing code from large-scale codebases (Liu et al., 2021a, 2022; Hu et al., 2023).

Early code search approaches (Satter and Sakib, 2016; Yang and Huang, 2017) relied heavily on Information Retrieval (IR) approaches for performing the keywords-based search. These approaches only work well if code and query share enough number of the same keywords. However, it is common that code and query are written in different keywords. The semantic gap between programming languages (e.g., Java) and natural languages (e.g., English) can cause a mismatch of keywords, leading to unsatisfactory effectiveness in code search (Rahman et al., 2019; Gu et al., 2022).

Neural networks have been frequently employed for code search to overcome the semantic gap as Deep Learning (DL) technologies have advanced (Liu et al., 2023; Yu et al., 2023). Gu et al. (2018) proposed a DL-based model named DeepCS that utilizes Long Short-Term Memory (LSTM) to jointly embed both code and query into a shared vector space, enabling a code search through vector similarity calculation. To reduce the computation complexity of DeepCS, UNIF (Cambronero

☆ Editor: Aldeida Aleti.

* Corresponding author at: School of Big Data and Software Engineering, Chongqing University, Chongqing, China.
E-mail addresses: zy.deng@cqu.edu.cn (Z. Deng), xuling@cqu.edu.cn (L. Xu), liu.chao@cqu.edu.cn (C. Liu), lhuangfu@sdsu.edu (L. Huangfu), mengy@cqu.edu.cn (M. Yan).

[1] https://www.github.com/.
[2] https://stackoverflow.com/.
[3] https://about.gitlab.com/.

https://doi.org/10.1016/j.jss.2023.111856
Received 30 March 2023; Received in revised form 1 August 2023; Accepted 25 September 2023
Available online 29 September 2023

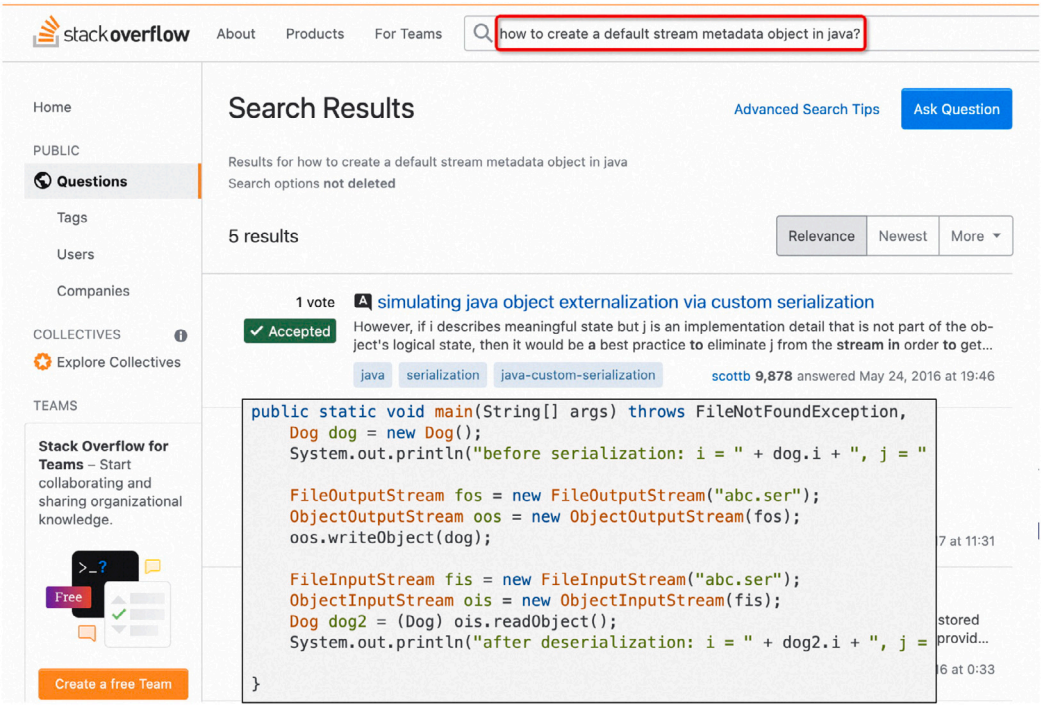

Fig. 1. Real world query example in stack overflow.

et al., 2019) substitutes the sophisticated embedding modules with a plain neural network called fastText and enhances performance with the attention mechanism. Haldar et al. (2020) noted that DeepCS neglects the structural semantics of the code and proposed CAT, which combines code text and Abstract Syntax Tree (AST) to enhance code representation. Recently, Xu et al. (2021) observed that the joint embedding could not fully capture the semantic correlations between code and query. They suggested using the co-attention mechanism (Bazmi et al., 2023; He et al., 2023) in the model called TabCS to understand the union semantics between code and query.

Existing Challenge and Empirical Study. Generally, the natural language description indicates the code's functionality. DL-based approaches embed codes and descriptions into a shared high-dimensional vector space and learn their matching semantic relations. Existing state-of-the-art approaches concentrate on exploiting multi-granularity code features to learn better representations, such as method name, code tokens, and API sequence (Gu et al., 2018; Haldar et al., 2020; Gu et al., 2021; Xu et al., 2021). However, code is an implementation of a low-level programming intent that does not have clear and high-level semantics as the query has (natural language description), which leads to inadequate semantic matching. Fig. 1 shows a failure query in Stack Overflow. The code provided in the first answer has only a few keywords that match the query, such as "stream" and "object". And the code also lost important semantic keywords, such as "create", "default", and "metadata". Obviously, this code is not the answer we expect to obtain.

In our study, we observed that the code snippets with similar syntax have similar natural language descriptions. This finding motivated us to investigate whether the descriptions of the most similar codes can be incorporated into original code tokens to achieve code semantic enrichment, since this description and the query are both in natural language. To validate our motivation, we performed a large-scale empirical study on a dataset comprising of over 2.2 million Java code-description pairs. We found that (1) in 63.22% of the samples, the descriptions of the code snippets with similar syntax contain more than one of the same functional keywords, and that (2) the code contains less semantic information than the description of its most similar code. These findings imply that the semantics of a code can be enriched by incorporating the description of the syntactically similar code snippet.

The Proposed Approach. This paper introduces a code semantic enrichment approach named **SemEnr** for deep code search. Specifically, SemEnr is comprised of two modules: a code enrichment module and a code search module. In the code enrichment module, we first enriched semantics for all code snippets in both training and testing data. For each code snippet, we retrieved the most similar code snippet from the training data. Then we estimate their syntactic similarity by utilizing Lucene, a widely employed text search engine (Hatcher and Gospodnetic, 2004). Thereafter, the semantics of the code snippet are represented by its code tokens and the description from its most similar code. The code search module contains a training phase and a testing phase. We used the attention mechanism during the training phase to embed the enriched code and query into a high-dimensional vector space. Second, we leveraged a multi-perspective mechanism to learn the local semantic correlations between code and query via row/column-wise Convolutional Neural Networks (CNNs). Generally, the semantic correlations enable SemEnr to learn better representations of code and query. During the testing phase, SemEnr compared the vector similarity of the input query to all enriched code from the codebase, and returned the top 10 code snippets ranked by similarity.

We conducted experiments on two extensively used Java datasets: *DeepCom-Java* (Hu et al., 2020) and *CSN-Java* (Husain et al., 2019), which contains more than 485k and 405k pairs of code and description, respectively. SemEnr achieved a Mean Reciprocal Rank (MRR) of 0.698 and 0.631, respectively, which outperforms the current advanced baseline models such as DeepCS (Gu et al., 2018), UNIF (Cambronero et al., 2019), CAT (Haldar et al., 2020), and TabCS (Xu et al., 2021). Moreover, we conducted a user study involving 50 real-world queries that Gu et al. (2018) obtained from Stack Overflow. The first relevant code snippets returned by SemEnr were ranked 5.52 on average. According to the Wilcoxon signed rank test (Wilcoxon, 1992), this improvement is statistically significant. These results indicate the usefulness of SemEnr over baselines in practical scenarios.

To summarize, we can outline our primary contributions as follows:

- **Empirical results:** Our research found that the code semantics can be enriched by incorporating the code tokens with the description of its most similar code. This finding has the potential to bridge the semantic gap that exists between queries and code.
- **A novel approach for code search:** We propose a novel code semantic enrichment approach, namely SemEnr, for deep code search. It aligns the semantics of code and query by enriching code with the description of its most similar code.
- **Performance evaluation:** We performed experiments on two widely used Java datasets and 50 real-world queries to evaluate our approach. Based on the experimental results, SemEnr outperforms the state-of-the-art approaches in terms of overall performance.

The structure of this article is presented in the following outline. Section 2 provides an overview of the background related to code search. Section 3 introduces a motivating example for SemEnr. Section 4 describes the empirical study to verify our observation. Section 5 presents an overview of SemEnr. Section 6 describes the experimental setup. Sections 7 and 8 present the experimental results and discussion, respectively. Section 9 is an introduction to related work. Finally, Section 10 summarizes our work and outlines possibilities for future research.

2. Background

The background of the code search task involved two aspects — deep learning techniques for code search and embedding techniques for code search.

2.1. Deep learning techniques for code search

Existing DL-based code search models typically learn vector representations of queries and candidate codes, and then perform the search process based on their similarity scores. These models utilize different neural networks for vector embedding, such as DeepCS (Gu et al., 2018) using LSTM and CARLCS-CNN (Shuai et al., 2020) using Convolutional Neural Network (CNN). LSTM (Sundermeyer et al., 2012) has the ability to model long-term dependencies on text sequences and memorize key information in text sequences, thus accurately representing vectors. CNN (Yin et al., 2017) extracts text features of different dimensions through convolutional kernels of different sizes, resulting in rich text features.

Currently, some advanced models use attention mechanisms to replace complex neural networks for vector representation. For example, models such as UNIF (Cambronero et al., 2019) and TabCS (Xu et al., 2021) are all built based on attention mechanisms. The main idea of the attention mechanism is to focus more attention on the part that contributes to the text by calculating the weight of each word, so as to obtain an accurate vector representation. Specifically, the attention mechanism can be divided into the following steps:

(a) Through a neural network layer, calculate the importance of each word in the text, and get the weight of each word.

(b) Based on the weight of each word and the word embedding vector, the weighted average value is calculated to obtain the vector representation of the text.

2.2. Embedding techniques for code search

Code Embedding. In DL-based code search models, multiple features of the code, including method name, API sequence, code tokens, and AST sequence, are typically utilized.

First, each feature is encoded into a vector matrix through an individual word embedding layer. Then, each feature matrix is further embedded by a neural network model. Finally, the outputs of four features are fused into a final code vector. In DeepCS (Gu et al., 2018), only the first three features are utilized, as Eq. (1), and the code features are processed through two LSTM models and a common Multi-Layer Perceptron (MLP), respectively. In TabCS (Xu et al., 2021), as Eq. (2), the AST is introduced as a code feature and attention mechanisms are used for feature processing.

$$v_{code} = M(L_1(E(s_{name})) + L_2(E(s_{api})) + M(E(s_{token}))) \tag{1}$$

$$v_{code} = Att_1(E(s_{name})) + Att_2(E(s_{api})) + Att_3(E(s_{token})) + Att_4(E(s_{AST})) \tag{2}$$

where E represents a word embedding layer, and L, M, and Att refers to the $LSTM$, MLP, and attention mechanism, respectively. s_{name}, s_{api}, s_{token}, and s_{AST} represent the features mentioned above.

Query Embedding. As for the query, the tokens are treated as the query feature. The query tokens are embedded and mapped into a vector matrix by an embedding layer and a neural network, respectively. In DeepCS and TabCS, the query is further embedded by a bidirectional LSTM and an attention mechanism, respectively.

$$v_{query} = BiLSTM\left(E\left(s_{query}\right)\right) \tag{3}$$

$$v_{query} = Att_5\left(E\left(s_{query}\right)\right) \tag{4}$$

where $BiLSTM$ refers to the $LSTM$ modules, and s_{query} represents query tokens.

Code Search. Generally, when obtaining the vectors of code and query, the DL-based models perform code search by computing the $cosine$ similarities between v_{code} and v_{query}.

$$cosine = \frac{v_{code} \cdot v_{query}}{\left\|v_{code}\right\| \cdot \left\|v_{query}\right\|} \tag{5}$$

The similarity score represents the correlation between a code-query pair. According to cosine similarities, the models return a list of codes in descending order.

3. Motivating examples

We present an instance that illustrates the semantic gap between code and description, and then discuss the observation motivating SemEnr.

The reason for using the code snippet descriptions as query texts to train models in most studies (Gu et al., 2018) is that there is currently no significant corpus available that includes pairs of human-written questions and their corresponding code snippets. In our work, we also treated descriptions as query texts. The description of the code often contains keywords that reflect the code functionality, such as the keywords “default”, “stream”, and “metadata” in the description of *covertStreamMetadata* method (as shown in Fig. 2). These keywords have semantic matching relationships with the method body. However, no code semantics share the same meaning as the other keywords “creates” and “merges” in the description, which exposes the semantic gap. The aim of the code search is to achieve a semantic match between the description and the code, and an inadequate semantic match will affect the search success rate in the code search task.

Fortunately, we found that similar code snippets have similar descriptions. For instance, we retrieved the most similar code using the Lucene engine based on the code tokens of *covertStreamMetadata*. We then obtained the most similar code snippet *covertImageMetadata* (as shown in Fig. 2) from the codebase, and its description contains not only the previously matched semantic keywords “default”, “stream”, and “metadata”, but also the mismatched semantic keywords “creates” and “merges”. Such results indicate that the code semantics can be enriched by incorporating the description of the most similar code into the code tokens, thus improving the matching of the semantics of code and description in code search.

```
# Description: creates a default stream metadata object and merges in the supplied metadata.
# Code:
   public IIOMetadata convertStreamMetadata (IIOMetadata inData,
                                             ImageWriteParam param) {
      IIOMetadata sm = getDefaultStreamMetadata(param);
      convertMetadata(STREAM_METADATA_NAME, inData, sm);
      return sm;
   }

Similar Code:
# Description: creates a default image metadata object and merges in the supplied metadata.
# Simalr Code:
   public IIOMetadata convertImageMetadata (IIOMetadata inData,
                                            ImageTypeSpecifier imageType,
                                            ImageWriteParam param) {
      GIFWritableImageMetadata im =
                (GIFWritableImageMetadata) getDefaultImageMetadata(imageType,
param);
      convertMetadata(IMAGE_METADATA_NAME, inData, im);
      return im;
   }
```

Fig. 2. The example of *covertStreamMetadata* and its similar code snippet *covertImageMetadata*.

4. Empirical study

In response to our observation, we carried out an empirical study to investigate whether this finding is widespread among large-scale open-source projects. In this section, we introduce the dataset used in the empirical study, as well as the statistical analysis and results.

4.1. Data collection and processing

To facilitate our statistical analysis, we used the large-scale Java datasets *Java-Large* collected by Alon et al. (2018), which contains approximately 16 million examples. This dataset contains the most popular 9500 Java projects on GitHub that have been established since January 2007. We observed that certain descriptions in the dataset had irrelevant content, such as an external resource link "http://...". As a result, we filtered the samples to enhance the dataset's quality as follows.

- Removed embedded comments from the code to avoid confusion between natural language and programming language in the code body.
- Removed examples with codes that cannot be parsed into an abstract syntax tree to reduce the number of code snippets that cannot be compiled.
- Removed examples that the number of description words is more than 256 or less than 3. Too many words will increase the risk of invalid information, and too few words will lead to semantic loss.
- Removed examples with descriptions containing special tokens (e.g. <img...> or https:...) to reduce the number of irrelevant description.
- Removed examples with non-English descriptions.

We also removed the stop words from the descriptions and retained only functional keywords, such as verbs and nouns, because these words reflect the semantics of the description. The final dataset contained more than 2.2 million pairs of Java methods and descriptions. For all pairs in the dataset, we split camel case for methods and changed them to lower case. Next, we utilized the Lucene search engine to obtain the code snippet that was most similar based on the tokens of each method. and extracted the description. For the convenience of subsequent writing, the phrase of *"similar description"* represents the description of the most similar code snippet. As a result, we obtained more than 2.2 million samples, and each sample contained an original code snippet, an original description, a similar code snippet, and a similar description.

4.2. Statistical analysis

***Analysis 1:* Do the descriptions of the code snippets with similar syntax share the same semantics?** This analysis's objective is to investigate the semantic relationship that exists between various descriptions of code snippets. In the event that the descriptions of similar code snippets share the same code semantics as the keywords used, the incorporation of the description of its similar code could help to clarify the semantics of a code snippet.

(1) Result: We statistically analyzed the number of the same keywords between the original descriptions and their similar descriptions. If the number was greater than three, we considered the two descriptions to be similar. Generally, when two descriptions have the same keywords, it indicates that they are semantically similar. In addition, we also analyzed the number of similar descriptions from the same project as the original description.

The results are illustrated in Table 1, in total 2,226,580 samples, the average length of the code descriptions is 9.17. So, we only analyze three similarity degrees: the number of the same keywords is greater than 3, 5 and 10. Specifically, we can observe that out of 1,407,649 samples, accounting for 63.22%, the descriptions and their similar descriptions share more than 3 same keywords. In 905,256 samples, accounting for 40.66%, the number of the same keywords is greater than 5. And in 358,709 samples, accounting for 16.11%, the number of the same keywords is greater than 10. The above numbers prove that the descriptions of the code snippets with similar syntax share the same functional keywords in 63.22% of all the samples. Moreover, in 16.11% of all samples, the original descriptions and similar descriptions showed high similarities, containing more than 10 of the same keywords that exceeded the descriptions' average length. Additionally, among these three statistical similarity degrees, there are

Table 1
The frequencies of the same keywords in similar code's descriptions.

	Number	Proportion
Total number of samples	2,226,580	–
Average length of descriptions	9.17	–
# The number of same keywords are greater than 3	1,407,649	63.22%
# Come from the same project	688,335	–
# The number of same keywords are greater than 5	905,256	40.66%
# Come from the same project	595,752	–
# The number of same keywords are greater than 10	358,709	16.11%
# Come from the same project	306,411	–

∗ 'Number' represents the number of samples that match the similarity degree of the corresponding row.
∗ 'Proportion' represents the proportion of samples corresponding row to the total.

48.90% (688,335/1,407,649), 65.81% (595,752/905,256), and 85.42% (306,411/358,709) samples that come from the same project as the original description, respectively. These statistics show that a considerable number of samples' similar descriptions and original descriptions come from the same project. However, most of the samples and original descriptions come from different projects. It is enough to motivate us to explore semantic information from the description of similar code snippets, regardless of whether it comes from the same project as the original description or not.

(2) Finding: In 63.22% of the samples, the descriptions of the code snippets with similar syntax share more than three functional keywords, which implies that we can explore semantic information from the description of the similar code snippet.

***Analysis 2:* Can the semantics of a code be enriched by incorporating the description of the most similar code?** Since a code snippet may have many similar code snippets, this RQ aims to investigate the feasibility of code semantic enrichment by incorporating the description of the most similar one. If this approach is reasonable and viable in practical usage, the semantic gap between code and query can be aligned.

(1) Result: We all know that some functional keywords in the original description can often be found in code tokens. From RQ1, we know that the similar description contains the same functional keywords as the original description. Therefore, we have statistically analyzed the frequencies of keywords of two aspects of the original code snippet and the similar description. Table 2 shows that the total number of keywords in the original description is 19,104,363, in which 21.66% of the keywords appear in the original code tokens, while 63.51% of the keywords appear in the similar descriptions. These data indicate that the similar description contains more semantic information than the original code (63.51% vs. 21.66%). In particular, the keywords that only appear in the original code tokens only account for 6.56% (1,248,148/19,104,363), while 48.40% (9,209,244/19,104,363) of the keywords only appear in the similar descriptions. These numbers demonstrate that the original code and the similar descriptions contain unique keywords of the original descriptions, respectively. These results indicate that the semantic gap between original code and original description can be connected by incorporating the description of the most similar code.

(2) Finding: The code itself contains less semantic information than the description of its most similar code (21.66% vs. 63.51%), which implies that we can enrich the semantics of a code by incorporating the description of the most similar code.

Table 2
The frequencies of the keywords in original descriptions under original code and similar description.

	Number	Proportion
Number of original description's keywords	19,104,363	–
# Appear in original code tokens	4,122,053	21.66%
# but not in similar descriptions	1,248,148	6.56%
# Appear in similar descriptions	12,083,149	63.51%
# but not in original code tokens	9,209,244	48.40%

∗ 'Number' represents the number of keywords corresponding row.
∗ 'Proportion' represents the proportion of keywords corresponding row to the total.

5. Our approach

In this work, we propose SemEnr, a code semantic enrichment approach for deep code search. Fig. 3 provides an overview of SemEnr and shows that it mainly includes two modules: code enrichment and code search. The code enrichment module contains a retrieval base, which is based on the training dataset, that includes the code snippets and their descriptions. For all code snippets in both the training and testing sets, we utilized the Lucene, an open-source search engine, with the goal to retrieve the code snippet that was most similar based on the input code tokens. For all code snippets in both the training and testing sets, we utilized the open-source search engine Lucene to retrieve the code snippet that was most similar based on the input code tokens. Thus, the semantics of the input code snippet are represented by its own code tokens and the description of the most similar code snippet. During the training phase of the code search module, three separately designed attention networks are used to embed code tokens, similar descriptions, and query descriptions to obtain code/description feature matrices. Thereafter, each feature matrix is fed into a multi-perspective co-attention mechanism separately to obtain the semantic correlations between the different code features and queries, which are used to enhance the code/description representative vector. Finally, SemEnr calculates the cosine similarity between the two vectors. During the testing phase, SemEnr compares the vector similarity of the input query to all the code in the codebase and returns the top 10 code snippets ranked by similarity.

5.1. Code enrichment module

Incorporating the description of the most similar code enriches the semantics of a code, as demonstrated by the statistical analysis in Section 4. Therefore, the code enrichment module aims to design an effective source code retrieval component based on the pairs of code snippets and descriptions.

Inspired by previous works (Yang et al., 2022), we constructed a retrieval module that estimates the relevance of code snippets by utilizing the BM25 (Askari et al., 2023) similarity measure to assess similarity in token. The bag-of-words retrieval algorithm BM25 has demonstrated its robustness and widespread adoption in practice. When a target code snippet is used as a query, the BM25 algorithm employs TF-IDF to calculate the frequency of each term within a document and adjusts it based on the inverse document frequency of the term. A higher BM25 score indicates greater similarity and relevance between the two code snippets. We leveraged the search engine Lucene (Hatcher and Gospodnetic, 2004) and use the training set as the retrieval base to simplify the retrieval component.

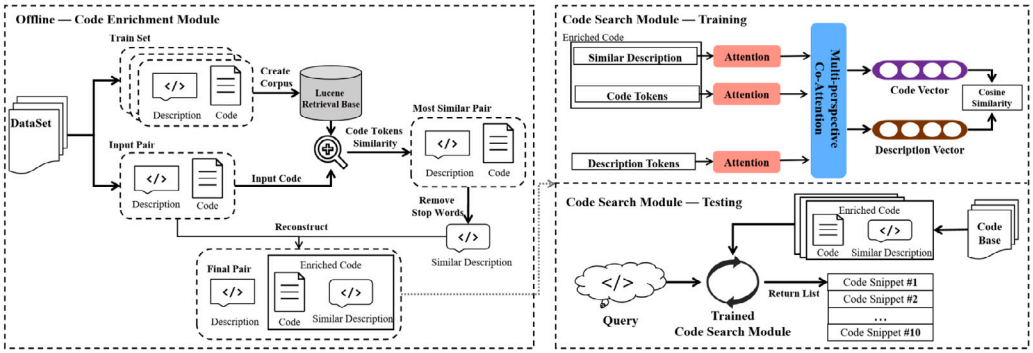

Fig. 3. Overview of SemEnr. * The Code Enrichment Module can be performed offline, using the code-description pairs of the training set in the dataset as the retrieval base for Lucene. Then, input a code-description pair, and retrieve the most similar code-description pair based on the code tokens. Finally, combine similar description with the input code to build the enriched code. * Other baseline models can be trained by replacing the Code Search Module in SemEnr and utilizing code-description pairs that are already enriched offline.

Fig. 3 shows the Lucene-based retrieval component returning the most similar code-description pair based on their code tokens' similarity. Thereafter, we extracted the description from the most similar pairs and combined this similar description with the input code snippet as the enriched code. Thus, we could obtain the reconstructed pair that contains the input description and the enriched code. We could run code enrichment module offline to obtain the reconstructed pairs of all pairs in the dataset, which were then used to train and test the subsequent code search module.

5.2. Code search module

Unlike the existing DL-based approaches (Gu et al., 2018; Haldar et al., 2020; Gu et al., 2021; Xu et al., 2021), we trained the code search module with the reconstructed pairs of enriched code and description instead of the original pairs. Specifically, the module learns the semantic matching relation between the enriched code and description through three steps: attention representation, multi-perspective co-attention, and joint embedding.

Step-1: Attention Representation. This step aims to obtain the embedded representation of code features and description features.

(1) Code features representation. The code features of the enriched code contain code tokens and similar descriptions. The textual semantics are reflected in code tokens extracted from code snippets using a lexical analyzer,[4] which is also frequently used in other works (Gu et al., 2018; Cambronero et al., 2019; Haldar et al., 2020; Xu et al., 2021). In addition, the similar description obtained from the code enrichment module contains some unique semantics.

Let $t = (t_1, t_2, \ldots, t_n)$ be a sequence of code tokens and $sd = (sd_1, sd_2, \ldots, sd_s)$ be a sequence of words of the similar description. We embedded each word as a vector for each feature by building a matrix $E \in R^{g*k}$ for initial embedding. The matrix contains k-Dimension vectors corresponding g words in the vocabulary. As a result, each feature can be randomly initialized to a feature matrix made up of a collection of word vectors.

Consider the code tokens, for instance, and assume that each word in the sequence of code tokens is represented by an initial word vector $t_i \in R^k$ of k-dimensions. To calculate the attention weight for each t_i, Eq. (6) is utilized, in which W and b are trainable parameters for the weight and bias, respectively. Next, the word vectors were multiplied by their respective attention weights and the resulting weighted vectors were concatenated together to create the code token feature matrix. The computation can be expressed as Eq. (7), where $T \in R^{k*n}$ represents the feature matrix of the code tokens and $\bigoplus$ denotes the concatenation operator. As for the similar description keywords, we applied the same formulas as for the code tokens to obtain the feature matrix $SD \in R^{k*s}$ as shown in Eqs. (8) and (9).

$$\alpha_{t_i} = SoftMax\left(tanh\left(W t_i + b\right)\right) \tag{6}$$

$$T = \alpha_{t_1} t_1 \bigoplus \alpha_{t_2} t_2 \bigoplus \ldots \bigoplus \alpha_{t_n} t_n \tag{7}$$

$$\alpha_{sd_i} = SoftMax\left(tanh\left(W sd_i + b\right)\right) \tag{8}$$

$$SD = \alpha_{sd_1} sd_1 \bigoplus \alpha_{sd_2} sd_2 \bigoplus \ldots \bigoplus \alpha_{sd_n} sd_n \tag{9}$$

(2) Description features representation. Natural language descriptions are typically made up of keywords that express the developer's intent for code functionality. We employed an attention mechanism to acquire profound semantic information from the code descriptions.

Suppose we have a description represented by $D = \{d_1, d_2, \ldots, d_o\}$, where $d_i \in R^k$ denotes the k-dimensional vector for the ith word in the description. Eqs. (10) and (11) outline the procedure for feature extraction, with the resulting final description feature matrix denoted as $D \in R^{k*o}$.

$$\alpha_{d_i} = SoftMax\left(tanh\left(W d_i + b\right)\right) \tag{10}$$

$$D = \alpha_{d_1} d_1 \bigoplus \alpha_{d_2} d_2 \bigoplus \ldots \bigoplus \alpha_{d_o} d_o \tag{11}$$

Step-2: Multi-perspective Co-Attention. We proposed a multi-perspective co-attention mechanism after the attention representation step to align the natural language description with each code feature. Specifically, this mechanism captures semantic associations with descriptions from two perspectives of code tokens and similar descriptions. The central concept of this mechanism is to combine these semantic associations, which helps to enrich the representation of code and descriptions, respectively, and subsequently benefits the semantic matching between code and description. The workflow of the multi-perspective co-attention mechanism is depicted in Fig. 4, with detailed process can be divided into three distinct sections, namely computing semantic association, calculating semantic vectors, and attention fusion.

(1) Computing Semantic Association. We obtained the code feature matrices T, SD, and description feature matrix D in the attention representation step. To enhance the code feature and description feature representations, we first established a matrix of mutually associated semantics between each of them. Taking the code token matrix T and description matrix D as instances, we computed a semantic association matrix $Co \in R^{n*o}$, where n denotes the number of code tokens, and o signifies the number of description words. The calculation of the semantic association matrix is given by Eq. (12), where the trainable parameter matrix $U \in R^{k*k}$ is used to combine the code tokens matrix and description matrix. To restrict the value of each matrix element to

[4] https://www.nltk.org/api/nltk.tokenize.html.

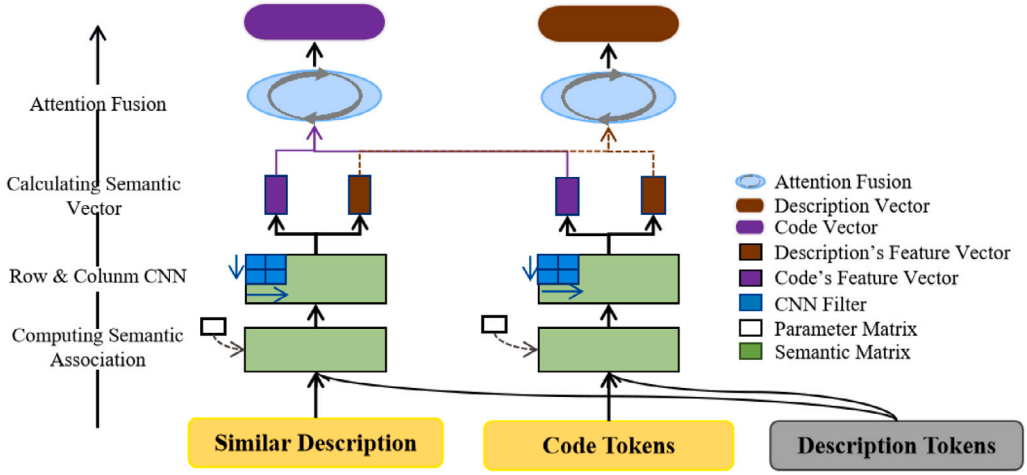

Fig. 4. Workflow of the multi-perspective co-attention mechanism.

fall within the range of −1 to 1, we applied the hyperbolic tangent *tanh* activation function.

$$Co = tanh\left(T^T U D\right) \tag{12}$$

Each element $co_{i,j}$ in the matrix Co represents the correlation between the sets of code token words and description words. To be specific, the ith row in the matrix Co corresponds to the correlation between each description word and the ith code word. In a similar way, the jth column in the matrix Co corresponds to the correlation between each code word and the jth description word.

(2) Calculating Semantic Vectors. CNNs are proficient at capturing contextual information within matrices and can efficiently extract abstract features from them (Kuttala et al., 2023). Therefore, we extracted the semantic matrix Co^T of the code tokens, and the semantic matrix Co^D of the descriptions with CNN $f(\cdot)$ along with the row and column directions of matrix Co, respectively, as shown in Eqs. (13)–(16). The equations utilize the filter $W \in R^{k*h}$ and specify a filter window size h of 2, where $co_{i,j}$ denotes an element within the semantic association matrix Co. And the FC refers to a fully connected layer, which assimilates the locally extracted information co_i^T and co_i^D acquired through convolution. Following information aggregation at the full connection layer, we used the average-pooling strategy to gather information to avoid the information omission problem of the max-pooling strategy.

$$co_i^T = f\left(W * co_{i:i+h-1} + b\right) \tag{13}$$

$$Co^T = ave - pooling\left(FC\left[co_1^T, co_2^T, \ldots, co_{n-h+1}^T\right]\right) \tag{14}$$

$$co_i^D = f\left(W * co_{i+h-1:i} + b\right) \tag{15}$$

$$Co^D = ave - pooling\left(FC\left[co_1^D, co_2^D, \ldots, co_{o-h+1}^D\right]\right) \tag{16}$$

Once the semantic matrices Co^T and Co^D were derived, we applied a softmax function to obtain the semantic weights $a^T \in R^n$ and $a^D \in R^o$, respectively. These weights were then employed to enhance the feature representations of their respective input. The process is illustrated as in Eqs. (17) and (18). Lastly, calculate the dot product of the feature matrix and semantic weight using Eq. (19) to obtain the semantic feature vector. V_{TD} represents the feature vector of code tokens semantically associated with the description feature matrix, and V_{DT} represents the feature vector of the descriptions semantically associated with the code tokens feature matrix.

$$a_i^T = \frac{exp\left(Co_i^T\right)}{\sum_{j=1}^{n} exp\left(Co_j^T\right)}, a_i^D = \frac{exp\left(Co_i^D\right)}{\sum_{j=1}^{o} exp\left(Co_j^D\right)} \tag{17}$$

$$a^T = \left[a_1^T, \ldots, a_n^T\right]^T, a^D = \left[a_1^D, \ldots, a_o^D\right]^T \tag{18}$$

$$V_{TD} = T \cdot a^T, V_{DT} = D \cdot a^D \tag{19}$$

Similarly, considering that the matching process for keywords is similar to that of code tokens, we performed the same processing operations. Thereafter, we could also obtain the similar description vector V_{SDD} and the corresponding description feature vector V_{DSD}.

(3) Attention Fusion. After calculating the semantic vectors, we obtained the code's feature vectors, $Code = \{V_{TD}, V_{SDD}\}$, and the description's feature vectors $Description = \{V_{DT}, V_{DSD}\}$. To account for varying degrees of importance among feature vectors, we utilized an attention mechanism between them to learn correlation weights, α_{v_c} and α_{v_d}, where $v_c \in Code$ and $v_d \in Description$. Subsequently, we executed a weighted fusion process to generate the ultimate code vector $V_C \in R^{k*c}$ and description vector $V_D \in R^{k*r}$, where c represents the total count of words across all code features, and r is twice the number of description words. Eqs. (20)–(23) demonstrate the calculation process, with W representing a parameter matrix that can be trained, along with a trainable bias term b.

$$\alpha_{v_c} = SoftMax\left(tanh\left(W v_c + b\right)\right) \tag{20}$$

$$V_C = \sum_{v_c \in Code} v_c \cdot \alpha_{v_c} \tag{21}$$

$$\alpha_{v_d} = SoftMax\left(tanh\left(W v_d + b\right)\right) \tag{22}$$

$$V_D = \sum_{v_d \in Description} v_d \cdot \alpha_{v_d} \tag{23}$$

Step-3: Joint Embedding. When the code and description embedding are completed, the code search module learns a joint embedding between V_C and V_D. Finally, each code snippet is ranked based on its cosine similarity to the relevant description. The cosine similarities are computed as in Eq. (24). In the testing phase of the code search module, given a query, the model recommends a list of codes in descending order based on the cosine similarity value.

$$cosine = (V_C \cdot V_D)/(\|V_C\| \cdot \|V_D\|) \tag{24}$$

5.3. Model optimization

In the experiment, we treated the natural language description of the code as an input query statement and a triple $\langle c, q^+, q^- \rangle$ is constructed to serve as each training instance. For each triple, a relevant query q^+ that provides an accurate description of c is selected from the

Table 3
Dataset statistics.

Datasets	# Training	# Validation	# Testing
DeepCom-Java	428,230	47,582	10,000
CSN-Java	356,303	39,590	10,000

query dataset, while an irrelevant query q^- that provides an inaccurate description of c is randomly chosen from the same dataset except q^+. Our approach assumes that if a code snippet and a query share similar semantics, their respective representation vectors in the vector space should be in close proximity to each other. To optimize the model, we use the margin ranking loss as shown in Eq. (25). The parameter θ denotes all of the model's parameters, and the default value for the margin constraint parameter β is set to 0.05. Specifically, our approach is designed to assign a higher similarity score to $S\left(c, q^+\right)$ as compared to $S\left(c, q^-\right)$. In the training process, the difference value between $S\left(c, q^+\right)$ and $S\left(c, q^-\right)$ would be closer to the margin β.

$$L(\theta) = \sum_{\langle c, q^+, q^- \rangle \in G} max\left(0, \beta - S\left(c, q^+\right) + S\left(c, q^-\right)\right) \tag{25}$$

We applied the Adam algorithm (Ogundokun et al., 2022) for training. With Adam optimization training our model, all trainable parameters are adjust following the gradient. Finally, SemEnr learns to obtain the final code representation vectors and query representation vectors.

6. Experiment setup

6.1. Dataset

We trained and tested SemEnr on two widely used large-scale Java datasets: *DeepCom-Java* and *CSN-Java*. The first *DeepCom-Java* dataset was collected by Hu et al. (2020) and contains over 485k code-description pairs. The other *CSN-Java* dataset was obtained from CodeSearchNet (Husain et al., 2019), and this dataset contains more than 405k Java code with descriptions. The two datasets were all collected from GitHub's Java repositories, and the descriptions were extracted from Javadoc. For fairness, we adopt the same dataset division method as the baseline researches (Gu et al., 2018; Xu et al., 2021; Shuai et al., 2020), as shown in Table 3. Randomly select 10k code-description pairs from the dataset as the testing set, and the remaining data is divided into the training set and validation set at a ratio of 9:1. It should be emphasized that in the code enrichment module, whether the training set, validation set or the testing set, the code semantic is enriched based on the retrieval base of the training set.

Moreover, we conducted a user study involving 50 real-world queries that Gu et al. (2018) obtained from Stack Overflow.

6.2. Implementation details

The implementation details of SemEnr are described as follows: The batch size is 128. The word embedding size for all word vectors is set to 100. We set training epochs to 400. We trained and tested SemEnr and baseline models in Python 3.6 with the Keras framework. It is worth noting that for the reproduction of the baseline models, we use the optimal parameter configuration set in their original paper to ensure the fairness of the experiments.

6.3. Baselines

DeepCS. Gu et al. (2018) presented this approach as the first DL-based code search model. The model utilizes LSTM and MLP to embed code snippets and their descriptions into a shared vector space. Code search is then performed by computing the similarity between the corresponding vectors.

UNIF. This is an attention-based model and was proposed by Cambronero et al. (2019). The model uses a neural network based on fastText and incorporates an attention mechanism to represent the code. For the embeddings of description tokens, a simple averaging technique is applied.

CAT. This is an AST-based model and was proposed by Haldar et al. (2020). This model uses a sequence encoder to integrate syntactic and semantic features, which are extracted from code tokens and their corresponding AST representations.

TabCS. This is an AST-based model, which is based on a two-stage attention mechanism, and was proposed by Xu et al. (2021). The attention mechanism is used in the first stage to extract semantics from code and queries. The second stage employs a co-attention mechanism to learn improved code/query representation by capturing their semantic association.

6.4. Evaluation metrics

To assess the performance of SemEnr, we employed two frequently used metrics: MRR and SuccessRate@k.

Mean Reciprocal Rank (MRR), which means the average of the reciprocal ranks in the top-10 of all queries. For the set of queries Q, the calculation of MRR: $MRR = (\sum_{q=1}^{|Q|} 1/FRank_q)/(|Q|)$, where $|Q|$ is the size of the Q set, and $FRank_q$ refers to the rank position of the ground-truth code for the qth query. Since developers tend to examine the top-ranked code snippets to find the desired code methods, we evaluated MRR only on the top-10 rated code snippets.

SuccessRate@k (SR@k), the percentage of queries for which the ground-truth code method is available in the top-k ranked list. For a query set Q, the calculation of SuccessRate@k: $(\sum_{q=1}^{|Q|} \delta\left(FRank_q \le k\right))/(|Q|)$. δ is an indicator function. If the top-k code snippets in the recommendation list include the ground-truth code of the ith query, it returns 1; Otherwise, it returns 0.

6.5. Research questions

We evaluate the performance of SemEnr and four baseline models for answering the following five research questions (RQs):

RQ1. How effective is SemEnr compared with baseline models?

To validate the suggested approach's performance, this question explores whether SemEnr outperforms the compared code search models such as DeepCS, UNIF, CAT, and TabCS on two commonly used datasets.

RQ2. How do different parts of SemEnr affect its performance?

In SemEnr, a code snippet is represented by code tokens and the description of its most similar code. We conducted feature ablation experiments to investigate how each feature affected the effectiveness of SemEnr. In addition to feature representation, we also explored the effect of the multi-perspective co-attention mechanism on model performance in this research.

RQ3. What is the impact of the CNN component on the effectiveness of SemEnr?

Table 4
Effectiveness comparison on DeepCom-JAVA dataset.

Model	SR@1	SR@5	SR@10	MRR
DeepCS	0.327	0.499	0.597	0.333
UNIF	0.529	0.682	0.772	0.525
CAT	0.596	0.749	0.814	0.582
TabCS	0.574	0.737	0.809	0.561
SemEnr	**0.720**	**0.811**	**0.853**	**0.698**

Table 5
Effectiveness comparison on CSN-JAVA dataset.

Model	SR@1	SR@5	SR@10	MRR
DeepCS	0.298	0.450	0.534	0.303
UNIF	0.493	0.629	0.690	0.398
CAT	0.538	0.674	0.737	0.531
TabCS	0.520	0.670	0.738	0.512
SemEnr	**0.648**	**0.749**	**0.795**	**0.631**

In SemEnr's multi-perspective co-attention mechanism, CNN component is important in the vector representation process of code and query. To verify the effectiveness of selecting CNN, we replaced CNN with LSTM for comparative experiments.

RQ4. Does the code enrichment module improve the performance of baseline models?

To explore the generalizability of the similar description keywords for the enhancement of the code search model, we introduced the code enrichment module into all baseline models to evaluate performance improvement.

RQ5. How is the computation efficiency of SemEnr?

To measure the efficiency of SemEnr, we compared the training time, testing time, parameters, and model sizes of all baseline models, and tested to evaluate whether SemEnr can substantially save more computation resources than the baseline models.

RQ6. How effective is SemEnr with real-world queries compared with the baseline models?

We assessed the real-world effectiveness of SemEnr by comparing its ability to retrieve relevant code snippets to that of DeepCS, UNIF, CAT, and TabCS. We conducted this comparison using 50 real-world queries collected by Gu et al.

7. Experiment results

7.1. RQ1: The effectiveness of SemEnr

Result: Tables 4–5 presents the experimental result of SemEnr compared to four baseline models (i.e, DeepCS, UNIF, CAT and TabCS) on two datasets. Among the four baseline models, the baseline CAT showed the best performance on the two datasets. However, SemEnr outperformed CAT by a substantial margin. For the *DeepCom-Java* dataset, SemEnr achieved an MRR of 0.698 and SR@1/5/10 with 0.720/0.811/0.823. SemEnr outperformed the state-of-the-art baseline CAT by 19.93% in terms of MRR, and by 20.81%/8.28%/4.79% in terms of SR@1/5/10. For the *CSN-Java* dataset, SemEnr achieved an MRR of 0.631 and SR@1/5/10 with 0.648/0.749/0.795. SemEnr also outperforms the state-of-the-art baseline CAT by 18.83% in terms of MRR, and by 20.45%/11.13%/7.87% in terms of SR@1/5/10.

Answer: The proposed model SemEnr outperformed all baseline models, DeepCS, UNIF, CAT, and TabCS.

Table 6
Effectiveness of three settings on DeepCom-Java dataset.

Model	SR@1	SR@5	SR@10	MRR
SemEnr (w/o SD)	0.569	0.730	0.803	0.558
SemEnr (w/o T)	0.589	0.661	0.696	0.574
SemEnr (w/o MC)	0.658	0.770	0.811	0.642
SemEnr	**0.720**	**0.811**	**0.853**	**0.698**

Table 7
Effectiveness of three settings on CSN-Java dataset.

Model	SR@1	SR@5	SR@10	MRR
SemEnr (w/o SD)	0.510	0.644	0.707	0.502
SemEnr (w/o T)	0.435	0.512	0.556	0.429
SemEnr (w/o MC)	0.580	0.695	0.742	0.570
SemEnr	**0.648**	**0.749**	**0.795**	**0.631**

7.2. RQ2: The influence of different parts on SemEnr

Result: We conducted an investigation into the relative importance of the features of the code, including code tokens and similar description. We removed one feature from SemEnr at a time. Moreover, in order to explore the effect of the multi-perspective co-attention mechanism on the validity of the approach, we removed this mechanism. In Tables 6–7, T, SD, and MC in SemEnr represent code tokens, similar descriptions and the multi-perspective co-attention mechanism, respectively.

In Tables 6–7, we can observe that the complete SemEnr achieved the best performance. Reducing either the code features or the multi-perspective co-attention mechanism significantly decreased model performance in SR@k and MRR on the two datasets. Specifically, on *DeepCom-Java* dataset, by removing the code tokens, similar description and multi-perspective co-attention mechanism, the MRR of SemEnr decreased by 17.77%, 20.06%, and 8.02%, respectively. And on CSN-Java dataset, the MRR of SemEnr decreased by 32.01%, 20.44%, and 9.67%, respectively.

These results indicate that the code tokens, the similar description, and the multi-perspective co-attention mechanism are indispensable in SemEnr. The code tokens are the main part of the code and contain the code textual semantics. Therefore, code tokens are essential during the phase of code representation. Moreover, the similar description keywords contain additional semantic information that may not be present in the code tokens, so the code tokens are used in combination with the similar description keywords to maximize the information of the code and improve the matching of corresponding queries. The multi-perspective co-attention mechanism is the main mechanism for the interaction between the code features and queries. It extracts the correlation between code tokens and queries and the correlation between similar description keywords and queries, respectively, to clearly match code snippets and queries.

Answer: The code semantic enrichment stage and the multi-perspective co-attention mechanism are both necessary to improve the effectiveness of SemEnr.

7.3. RQ3: The impact of CNN component

Result: To verify the effectiveness of selecting CNN in the multi-perspective co-attention mechanism, we constructed two comparative

Table 8
Effectiveness of different neural networks on DeepCom-Java dataset.

Model	SR@1	SR@5	SR@10	MRR
SemEnr (Avg-pooling)	0.681	0.787	0.831	0.665
SemEnr (LSTM)	0.700	0.793	0.839	0.684
SemEnr	**0.720**	**0.811**	**0.853**	**0.698**

Table 9
Effectiveness of different neural networks on CSN-Java dataset.

Model	SR@1	SR@5	SR@10	MRR
SemEnr (Avg-pooling)	0.583	0.694	0.748	0.572
SemEnr (LSTM)	0.603	0.712	0.763	0.591
SemEnr	**0.648**	**0.749**	**0.795**	**0.631**

Table 10
Effectiveness comparison of using similar description keywords in models on DeepCom-Java dataset.

Model	SR@1	SR@5	SR@10	MRR
DeepCS	0.327	0.499	0.597	0.333
DeepCS+SD	**0.440**	**0.593**	**0.670**	**0.431**
UNIF	0.529	0.682	0.772	0.525
UNIF+SD	**0.663**	**0.772**	**0.816**	**0.648**
CAT	0.596	0.749	0.814	0.582
CAT+SD	**0.691**	**0.802**	**0.846**	**0.673**
TabCS	0.574	0.737	0.809	0.561
TabCS+SD	**0.675**	**0.793**	**0.842**	**0.659**

Table 11
Effectiveness comparison of using similar description keywords in models on CSN-Java dataset.

Model	SR@1	SR@5	SR@10	MRR
DeepCS	0.298	0.450	0.534	0.303
DeepCS+SD	**0.344**	**0.495**	**0.574**	**0.345**
UNIF	0.493	0.629	0.690	0.398
UNIF+SD	**0.582**	**0.697**	**0.744**	**0.571**
CAT	0.538	0.674	0.737	0.531
CAT+SD	**0.586**	**0.702**	**0.752**	**0.574**
TabCS	0.520	0.670	0.738	0.512
TabCS+SD	**0.588**	**0.712**	**0.769**	**0.577**

models based on SemEnr, namely SemEnr-Avgpooling and SemEnr-LSTM. SemEnr-Avgpooling uses average pooling technology instead of CNN to aggregate information. SemEnr-LSTM uses LSTM to aggregate information.

In Tables 8–9, we can observe that the SemEnr achieved the best performance. For the *DeepCom-Java* dataset, SemEnr outperforms SemEnr-Avgpooling by 4.96% in terms of MRR and outperforms SemEnr-LSTM by 2.05% in terms of MRR. For the *CSN-Java* dataset, SemEnr also outperforms SemEnr-Avgpooling by 10.31% in terms of MRR and outperforms SemEnr-LSTM by 6.77% in terms of MRR.

Answer: SemEnr can further improve the effectiveness of code search by multi-perspective co-attention representation learning based on CNN embedding technology.

7.4. RQ4: The improvement of all baselines by adding a code enrichment module

Result: To explore the generalizability of the code enrichment module of SemEnr, we introduced it to all the baseline models for comparison on two datasets. In Tables 10–11, "+SD" represents that similar description keywords have been added to the basis of the original model. Specifically, the similar description retrieved in the code enrichment module is taken as a new code feature in the code representation phase of each model. For example, DeepCS+SD collectively represents code through original features used in DeepCS (i.e., method name, API sequence, and code tokens) and similar description.

For the *DeepCom-Java* dataset, the performance of all baseline models (i.e, DeepCS, UNIF, CAT and TabCS) improved by a substantial margin after introducing the SD feature, with improvements of 29.43%, 22.86%, 15.64%, and 17.47% in terms of MRR; 34.56%/18.84%/12.23%, 25.33%/13.20%/5.70%, 15.94%/7.08%/3.93%, and 17.60%/7.60%/4.08%, in terms of SR@1/5/10. For *CSN-Java* dataset, the improvements of all baseline models (i.e, DeepCS, UNIF, CAT and TabCS) after using SD feature are 13.86%, 43.47%, 8.10% and 11.27% in terms of MRR; 15.44%/10.00%/7.49%, 18.05%/10.81%/7.83%, 8.92%/4.15%/2.04% and 13.08%/6.27%/4.20% in terms of SR@1/5/10. These results suggest that the code search models can be generally improved by adding the similar description.

Answer: All the baseline models can be improved by incorporating the code enrichment module of our SemEnr.

Table 12
Resource cost for model training and testing on DeepCom-Java dataset.

Model	Training	Testing	Parameters	Size
DeepCS	83.3 h	1.08 s/query	33,431,100	131M
UNIF	2.8 h	0.23 s/query	7,617,150	30M
CAT	133.0 h	2.52 s/query	26,716,900	200M
TabCS	7.6 h	0.58 s/query	15,667,502	61M
SemEnr	8.2 h	0.25 s/query	10,671,522	42M

Table 13
Resource cost for model training and testing on CSN-Java dataset.

Model	Training	Testing	Parameters	Size
DeepCS	66.7 h	1.11 s/query	9,801,600	38M
UNIF	3.3 h	0.27 s/query	5,025,950	19M
CAT	100.0 h	2.53 s/query	18,934,900	74M
TabCS	5.6 h	0.49 s/query	8,039,602	31M
SemEnr	6.7 h	0.28 s/query	6,311,922	25M

7.5. RQ5: The efficiency of SemEnr

Result: Tables 12–13 present the results of comparison of the training time, testing time, parameters and model size of all baselines on two datasets. All models were evaluated on the same experimental setup.

For the *DeepCom-Java* dataset, the results show that DeepCS, UNIF, CAT, TabCS, and SemEnr required 83.3, 2.8, 133, 7.6, and 8.2 h for training, respectively, and spent approximately 1.08, 0.23, 2.52, 0.58, and 0.25 s with each query in the process of testing. Compared to DeepCS and CAT, SemEnr is 10 times and 16 times faster in model training.

For *CSN-Java* dataset, the results show that DeepCS, UNIF, CAT, TabCS and SemEnr take 66.7, 3.3, 100, 5.6 and 6.7 h for training, respectively, and spend about 1.11, 0.27, 2.53, 0.49 and 0.28 s for each query in the process of testing. Compared to DeepCS and CAT, SemEnr is 10 times and 15 times faster in model training.

On both datasets, SemEnr is second only to the simplest attention-based model UNIF in terms of testing efficiency, parameters and model size, because UNIF simply implements code search through independent embedding, without considering the correlation between code and query.

These results suggest that attention mechanism based models, such as TabCS and UNIF, perform well in terms of efficiency. Both DeepCS and CAT are known to be computationally inefficient, requiring significant time and computational resources to train due to their complex network structures and time-consuming optimization processes. In addition, they have large numbers of parameters and big model sizes, as they rely on CNN-based or LSTM-based models. SemEnr introduces a CNN-based multi-perspective attention mechanism to capture the fine-grained correlation between code and query. Although it has a slight

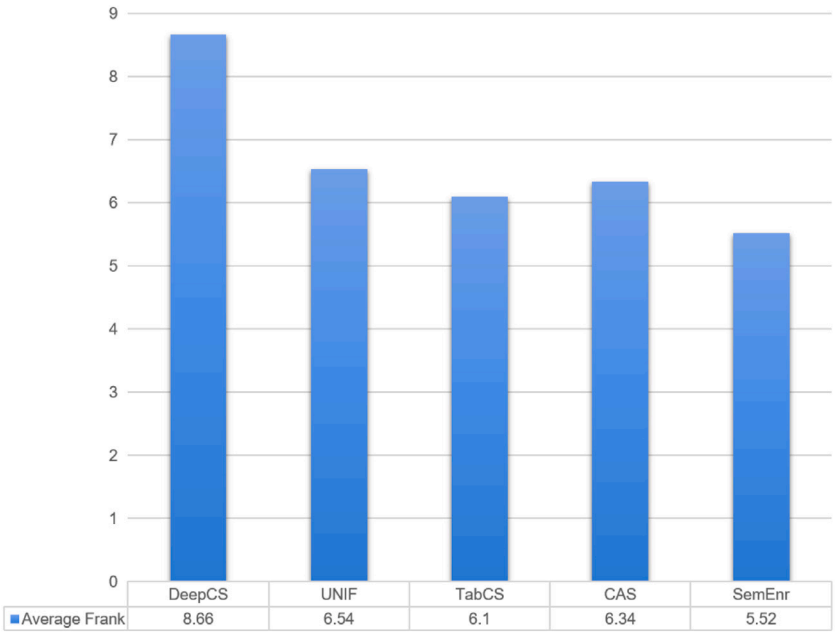

Fig. 5. Results of average Frank.

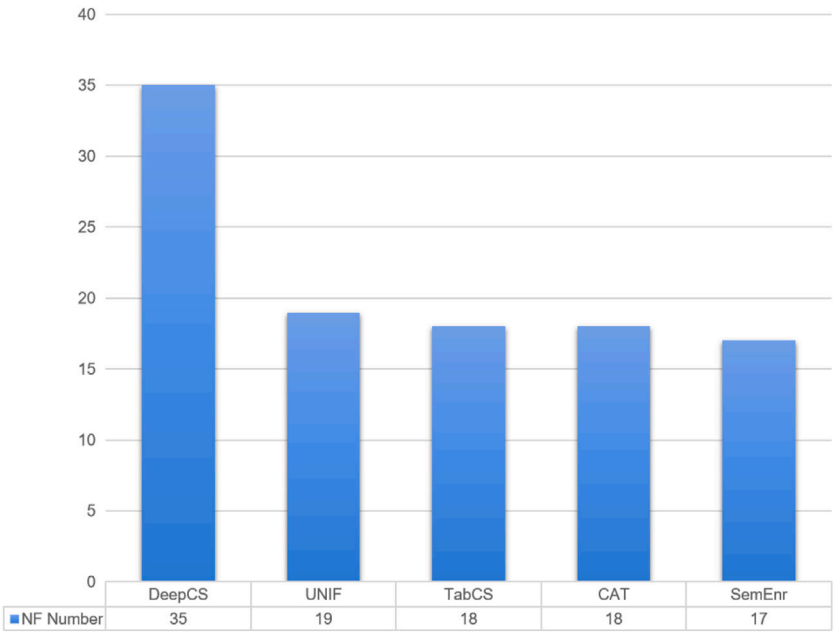

Fig. 6. Results of NF number.

impact on the efficiency of the model, we believe that it is worth exchanging a little efficiency loss for performance improvement.

Answer: SemEnr shows good performance on efficiency.

7.6. RQ6: The effectiveness of SemEnr on real-world queries

Result: Based on *DeepCom-Java*, we conducted a user study on SemEnr and its corresponding baselines using 50 real-world queries. The corresponding queries can be found in the work (Gu et al., 2018) of Gu et al. For each query, the code search model returns the top K results retrieved (K = 10). In the process of the experiment, two experienced developers manually identified the relevance of the results, and any different opinions were united through open discussions. 'NF' indicates that no relevant code snippet was found in the top K results. For queries marked as 'NF', we conservatively assigned the FRank value of 11. Figs. 5–6 shows the comparison of the models. We observed that SemEnr achieved more relevant results with an average FRank of 5.52 than DeepCS (8.66), UNIF (6.54), TabCS (6.10), and CAT (6.34). These correspond to reduced performance rates in FRank, which were as high as 36.3%, 15.6%, 9.5%, and 12.9%, respectively. 'NF Number' indicates how many of the 50 queries failed to find the relevant snippet, and SemEnr has the smallest corresponding value. The results show that our approach performed better than all the baselines on real queries. The retrieve module can generate more semantic information so that SemEnr can better understand the correlation between code and query.

To assess the statistical difference between SemEnr and the baselines, we conducted the Wilcoxon signed-rank test (Wilcoxon, 1992) on

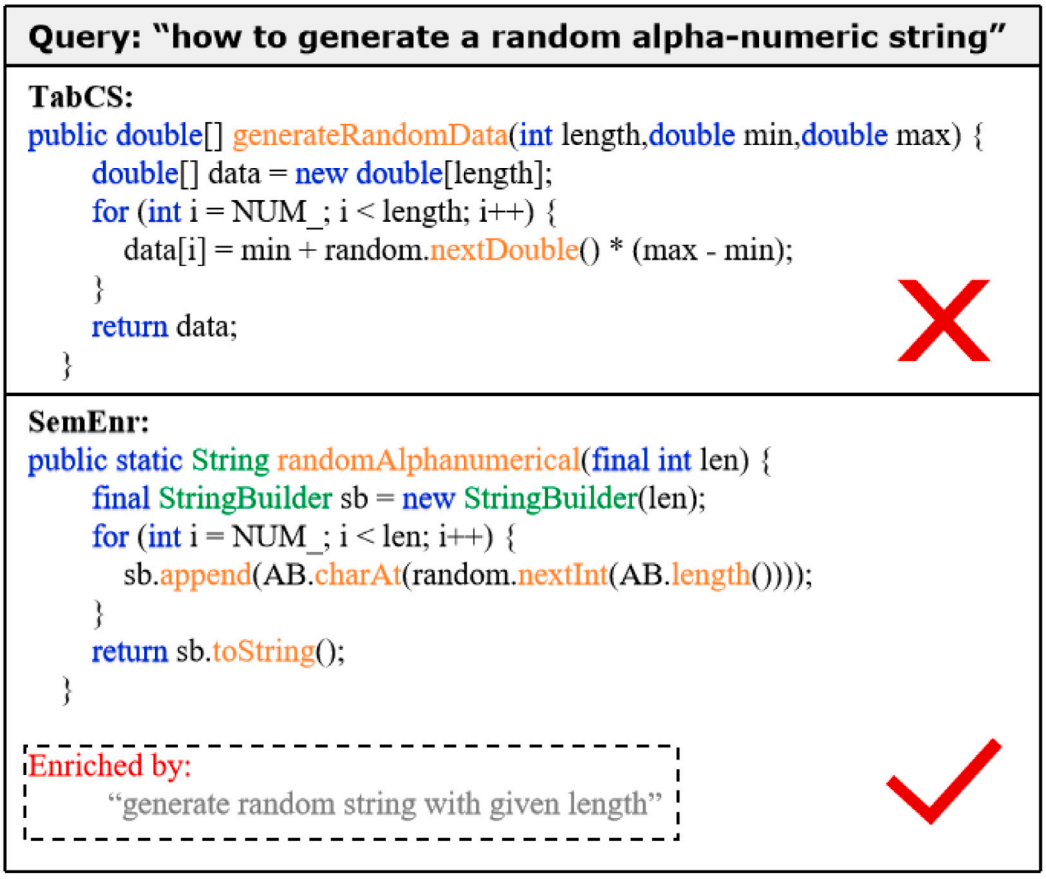

Fig. 7. The first code snippet returned by TabCS and SemEnr for the given query.

the FRank values at a significance level of 5%. The obtained p-value was less than 0.01, indicating that the improvements achieved by SemEnr in comparison to the baselines are statistically significant and substantial.

Answer: In terms of statistical significance for real-world queries, SemEnr outperforms baselines.

8. Discussion

8.1. Why Does SemEnr Work Well?

Based on the experimental results, it can be concluded that our approach is superior in effectiveness to existing state-of-the-art models. We analyzed the two possible reasons, including the code semantic enrichment and multi-perspective semantic representation.

Retrieval-based code semantic enrichment. From the results of large-scale statistical analysis in Section 4, we know that the descriptions of code snippets with similar semantics contain additional semantic information, which may be related to the query but is not reflected in the method body of the original code. Therefore, combining the code tokens with the description keywords of similar code can enhance the code semantics representation and better match the query and code.

Multi-perspective co-attention mechanism. This mechanism explores correlations between code and queries from different perspectives. From the perspective of code tokens, our approach learns the correlation between each token and each query word in an attempt to understand the semantic correlation between the code itself and the query. However, the semantic gap between programming languages and natural languages makes it difficult for our approach to understanding the correlation. Fortunately, from the perspective of the description of similar code, our approach also learns the correlation between each word in the similar description and each query word, and the similar description and query are natural languages with homology, so the model can capture the semantic correlation between them better. Therefore, under the multi-perspective co-attention mechanism, the similar code's description can enrich the semantic correlation representation, thus bridging the semantic gap between the code and the query, and improving the code search task.

We analyzed an example to understand the above advantages of SemEnr more intuitively. Fig. 7 shows the first code snippet returned by TabCS and SemEnr, respectively, based on the query "how to generate a random alpha-numeric string". TabCS returned an error code that reflected only part of the query semantics, such as "generate" and "random", resulting in an insufficient semantic matching between the code and query. Similarly, the correct code returned by SemEnr only reflects the query semantics of "random", "alpha-numeric", and "string", and lacks the semantics of "generate". However, SemEnr enriches code by incorporating the description of the most similar code, which contains the semantic keywords "generate", and captures semantic associations with the query from multiple perspectives. As a result, the code snippet returned by SemEnr can match this query better than that returned by TabCS.

8.2. Threats to validity

The validity of the proposed model SemEnr could be threatened by the following aspects. One is that our work focused on Java datasets only, and all findings and evaluation results are restricted to this domain, which does not necessarily mean that our approach will work in other language fields (e.g., C++, Golang, Python). Although, the principle of SemEnr is not limited to one specific programming language. However, constructing the necessary prior knowledge requires

another large-scale empirical analysis, and thus, this remains an area for future research. Another potential threat is the implementation of the baselines, as the baseline reproducing by us may cause deviation. To minimize this potential threat, we addressed it by re-running DeepCS and TabCS using the source code shared by the authors on GitHub. For models without public source code, such as UNIF and CAT, we reproduced the corresponding models strictly according to the original paper.

9. Related work

Source code search tasks aim to retrieve relevant code from a vast codebase based on the search query intent of developers. Initially, code search approaches primarily relied on technologies such as information retrieval and natural language processing, which consider textual dependencies between code and query. As an example, Sindhgatta (2006) introduced JSearch, a tool for source code search. By extracting various code elements, this tool indexes the source code and utilizes information retrieval techniques. Sourcerer, proposed by Bajracharya et al. (2007), is a tool based on Lucene (a conventional text search engine) that searches for code by analyzing similarities between text characteristics and code properties. CodeHow, proposed by Lv et al. (2015), enriches queries with related APIs. CodeMatcher, proposed by Liu et al. (2021b), understands irrelevant or noisy keywords and learns sequential relationships between query words and code words.

However, these traditional IR-based models tend to treat natural language and programming language as combinations of characters or words, without considering their semantic meanings.

To address the above issue, Gu et al. (2018) employed deep learning technology to learn semantic information and proposed DeepCS. It uses two separate LSTMs to generate high-dimensional vector representations of a natural language query and a code snippet, which are subsequently mapped to the same vector space. The similarity between the two vectors is then calculated using cosine similarity. Inspired by them, more studies have striven to improve code search tasks on their foundation. Cambronero et al. (2019) proposed an attention-based model UNIF, which is simpler compared to most DL-based code search model. Wan et al. (2019) proposed MMAN, which integrates multimodal representations by applying attention mechanisms. Eberhart and McMillan (2022) proposed a way to improve code search performance by using function names and comments to generate clarifying questions. Xu et al. (2021) proposed TabCS on the basis of the co-attention mechanism, which learns the interdependent representations for the embedded code and query and achieves significant performance improvement over the other models.

The above mentioned models mainly focused on developing a better representation method for code and query. In contrast, our approach aims to improve code search by enriching code semantics. The experimental results showed that the proposed model SemEnr outperforms the state-of-the-art baselines. In addition, the stage of code semantic enrichment in SemEnr is capable of enhancing the code search performance of the baselines.

10. Conclusion

In this paper, we conducted a large-scale empirical analysis on more than 2.2 million code-description pairs, and the key finding is that the semantics of a code can be enriched by incorporating the description of the code snippets with similar syntax. Our finding has inspired us to introduce a new method for enhancing deep code search, which we refer to as SemEnr and involves enriching the code semantics. SemEnr contains two modules. The code enrichment module retrieves the most similar code and extracts its description to enrich the semantics of the input code. The code search module learns the semantic correlations between code and query and enhances their representative vectors through the multi-perspective co-attention mechanism. We carried out comprehensive experiments on two frequently utilized Java datasets. The experimental results show that code search can be better performed after enriching code semantics, and the performance of SemEnr also exceeds the baseline models. All the source code and data of this study can be found at: https://github.com/cqu-isse/SemEnr.

In the future, we will expand SemEnr to the search field of other programming languages, such as c#, python, and GoLang, and continue to explore better semantic enrichment methods and vector representation techniques for code and query. Moreover, on the basis of this work, we will consider whether SemEnr could benefit from top-k similar code snippets in our future work.

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

https://github.com/cqu-isse/SemEnr.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (No. 62372071 and No. 62202074), the General Program of Chongqing Natural Science Foundation (cstc202ljcyj-msxmX0309), the Chongqing Technology Innovation and Application Development Project (No. CSTB2022TIAD-STX0007 and No. CSTB2022TIAD-KPX0067), the China Post-doctoral Science Foundation (2022M710519).

References

Alon, U., Brody, S., Levy, O., Yahav, E., 2018. code2seq: Generating sequences from structured representations of code. arXiv preprint arXiv:1808.01400.

Askari, A., Abolghasemi, A., Pasi, G., Kraaij, W., Verberne, S., 2023. Injecting the BM25 score as text improves BERT-based re-rankers. In: Advances in Information Retrieval: 45th European Conference on Information Retrieval, ECIR 2023, Dublin, Ireland, April 2–6, 2023, Proceedings, Part I. Springer, pp. 66–83.

Bajracharya, S., Ngo, T., Linstead, E., Rigor, P., Dou, Y., Baldi, P., Lopes, C., 2007. Sourcerer: a search engine for open source code. In: International Conference on Software Engineering. Citeseer.

Bazmi, P., Asadpour, M., Shakery, A., 2023. Multi-view co-attention network for fake news detection by modeling topic-specific user and news source credibility. Inf. Process. Manage. 60 (1), 103146.

Cambronero, J., Li, H., Kim, S., Sen, K., Chandra, S., 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.

Eberhart, Z., McMillan, C., 2022. Generating clarifying questions for query refinement in source code search. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, pp. 140–151.

Gu, W., Li, Z., Gao, C., Wang, C., Zhang, H., Xu, Z., Lyu, M.R., 2021. CRaDLe: Deep code retrieval based on semantic dependency learning. Neural Netw. 141, 385–394.

Gu, W., Wang, Y., Du, L., Zhang, H., Han, S., Zhang, D., Lyu, M.R., 2022. Accelerating code search with deep hashing and code classification. arXiv preprint arXiv:2203.15287.

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.

Haldar, R., Wu, L., Xiong, J., Hockenmaier, J., 2020. A multi-perspective architecture for semantic code search. arXiv preprint arXiv:2005.06980.

Hatcher, E., Gospodnetic, O., 2004. Lucene in Action (in Action Series). Manning Publications Co..

He, X., Wang, Y., Zhao, S., Chen, X., 2023. Co-attention fusion network for multimodal skin cancer diagnosis. Pattern Recognit. 133, 108990.

Hu, X., Li, G., Xia, X., Lo, D., Jin, Z., 2020. Deep code comment generation with hybrid lexical and syntactical information. Empir. Softw. Eng. 25 (3), 2179–2217.

Hu, F., Wang, Y., Du, L., Li, X., Zhang, H., Han, S., Zhang, D., 2023. Revisiting code search in a two-stage paradigm. In: Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining. pp. 994–1002.

Husain, H., Wu, H.-H., Gazit, T., Allamanis, M., Brockschmidt, M., 2019. Codesearchnet challenge: Evaluating the state of semantic code search. arXiv preprint arXiv:1909.09436.

Kuttala, R., Subramanian, R., Oruganti, V.R.M., 2023. Multimodal hierarchical CNN feature fusion for stress detection. IEEE Access.
Liu, Y., Li, S., Tilevich, E., 2022. Toward a better alignment between the research and practice of code search engines. In: 2022 29th Asia-Pacific Software Engineering Conference (APSEC). IEEE, pp. 219–228.
Liu, C., Xia, X., Lo, D., Gao, C., Yang, X., Grundy, J., 2021a. Opportunities and challenges in code search tools. ACM Comput. Surv. 54 (9), 1–40.
Liu, C., Xia, X., Lo, D., Liu, Z., Hassan, A.E., Li, S., 2021b. Codematcher: Searching code based on sequential semantics of important query words. ACM Trans. Softw. Eng. Methodol. (TOSEM) 31 (1), 1–37.
Liu, S., Xie, X., Siow, J., Ma, L., Meng, G., Liu, Y., 2023. GraphSearchNet: Enhancing GNNs via capturing global dependencies for semantic code search. IEEE Trans. Softw. Eng..
Lv, F., Zhang, H., Lou, J.-g., Wang, S., Zhang, D., Zhao, J., 2015. Codehow: Effective code search based on api understanding and extended boolean model (e). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, pp. 260–270.
Ogundokun, R.O., Maskeliunas, R., Misra, S., Damaševičius, R., 2022. Improved CNN based on batch normalization and adam optimizer. In: Computational Science and Its Applications–ICCSA 2022 Workshops: Malaga, Spain, July 4–7, 2022, Proceedings, Part V. Springer, pp. 593–604.
Rahman, M.M., Roy, C.K., Lo, D., 2019. Automatic query reformulation for code search using crowdsourced knowledge. Empir. Softw. Eng. 24 (4), 1869–1924.
Satter, A., Sakib, K., 2016. A search log mining based query expansion technique to improve effectiveness in code search. In: 2016 19th International Conference on Computer and Information Technology (ICCIT). IEEE, pp. 586–591.
Shuai, J., Xu, L., Liu, C., Yan, M., Xia, X., Lei, Y., 2020. Improving code search with co-attentive representation learning. In: Proceedings of the 28th International Conference on Program Comprehension. pp. 196–207.
Sindhgatta, R., 2006. Using an information retrieval system to retrieve source code samples. In: Proceedings of the 28th International Conference on Software Engineering. pp. 905–908.
Sundermeyer, M., Schlüter, R., Ney, H., 2012. LSTM neural networks for language modeling. In: Thirteenth Annual Conference of the International Speech Communication Association.
Wan, Y., Shu, J., Sui, Y., Xu, G., Zhao, Z., Wu, J., Yu, P., 2019. Multi-modal attention network learning for semantic source code retrieval. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, pp. 13–25.
Wilcoxon, F., 1992. Individual comparisons by ranking methods. In: Breakthroughs in Statistics. Springer, pp. 196–202.
Xu, L., Yang, H., Liu, C., Shuai, J., Yan, M., Lei, Y., Xu, Z., 2021. Two-stage attention-based model for code search with textual and structural features. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 342–353.
Yang, Y., Huang, Q., 2017. IECS: Intent-enforced code search via extended Boolean model. J. Intell. Fuzzy Systems 33 (4), 2565–2576.
Yang, Y., Xu, L., Yan, M., Xu, Z., Deng, Z., 2022. A naming pattern based approach for method name recommendation. In: 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE). IEEE, pp. 344–354.
Yin, W., Kann, K., Yu, M., Schütze, H., 2017. Comparative study of CNN and RNN for natural language processing. arXiv preprint arXiv:1702.01923.
Yu, D., Yang, Q., Chen, X., Chen, J., Xu, Y., 2023. Graph-based code semantics learning for efficient semantic code clone detection. Inf. Softw. Technol. 156, 107130.

Zhongyang Deng
Email: zy.deng@cqu.edu.cn.
Profession: Student.

Zhongyang Deng received his bachelor's degree in School of Big Data and Software Engineering, Chongqing University in 2017, and is currently studying for master's degree in School of Big Data and Software Engineering, Chongqing University.

Ling Xu
Email: xuling@cqu.edu.cn.
Profession: PhD, Associate Professor/Master's Supervisor.

Ling Xu is now an Associate Professor at the School of Big Data and Software Engineering, Chongqing University, China. She received Ph.D. degree in computer application in 2009 and spent a year of study at Georgia Institute of technology in 2017. Her Research interests include intelligent software engineering, natural language processing and Big data analytics.

Chao Liu
Email: liu.chao@cqu.edu.cn.
Profession: PhD, Associate Professor/Master's Supervisor.

Chao Liu received a Ph.D. in software engineering from Chongqing University, China, in 2018. He was a Post-Doctoral Research Fellow at Zhejiang University, China, from 2019 to 2021. He is currently an Associate Professor at Chongqing University, China. His research interests include intelligent software engineering, program analysis, and software reuse.

Luwen Huangfu
Email: lhuangfu@sdsu.edu.
Profession: PhD, Assistant Professor/Supervisor.

Luwen Huangfu received the B.S. degree in software engineering from Chongqing University, Chongqing, China, the M.S. degree in computer science from the Chinese Academy of Sciences, Beijing, China, and the Ph.D. degree in management information systems from University of Arizona, Tucson, AZ, USA. She is currently an Assistant Professor with Fowler College of Business, San Diego State University, San Diego, CA, USA, where she is also with the Center for Human Dynamics in the Mobile Age. She has authored/coauthored more than 30 scientific papers in venues, such as IEEE Transactions on Cybernetics, IJCAI, ACM MM, Journal of Medical Internet Research (JMIR), IEEE Transactions on Intelligent Transportation Systems, Pacific Asia Journal of the Association for Information Systems, IEEE Signal Processing Letters, LREC, ICASSP, and IEEE ISI. Her research interests include business analytics, text mining, data mining, software management, computer vision, artificial intelligence, and healthcare management.

Meng Yan
Email: mengy@cqu.edu.cn.
Profession: PhD, Doctoral Supervisor.

Meng Yan is now a Research Professor at the School of Big Data and Software Engineering, Chongqing University, China. He received Ph.D. degree in June 2017 under the supervision of Prof. Xiaohong Zhang from Chongqing University, China. His current research focuses on how to improve developers' productivity, how to improve software quality, and how to reduce the effort during software development by analyzing rich software repository data.