Multi-intent Description of Keyword Expansion for Code Search

Haize Hu[1](✉), Jianxun Liu[1], and Lin Xiao[2]

[1] Hunan University of Science and Technology, Xiangtan 411201, Hunan, China
hhz@mail.hnust.edu.cn
[2] Hunan Normal University, Changsha 410081, Hunan, China

Abstract. To address the issue of discrepancies between online query data and offline training data in code search research, we propose a novel code search model called multi intent description keyword extension-based code search (MDKE-CS). Our model utilizes offline training data to expand query data, thereby mitigating the impact of insufficient query data and intention differences between training and query data on search results. Furthermore, we construct a multi-intention description keyword vocabulary library based on developers, searchers, and discussants from the StackOverflow Q&A library to further expand the query. To evaluate the effectiveness of MDKE-CS in code search tasks, we conducted comparative experimental analyses using two baseline models, DeepCS and UNIF, as well as WordNet and BM25 extension methods. Our experimental results demonstrate that MDKE-CS outperforms the baseline models in terms of R@1, R@5, R@10, and MRR values.

Keywords: Code search · Multi-intention · Expand query

1 Introduction

With the increasing number of developers who upload and share their code fragments on open source communities, the code resources available in these communities have become increasingly abundant. [1] This continuous enrichment of open source community resources has provided a vital foundation for the development of code search [2]. By searching for existing code fragments in open source communities, software developers can modify and reuse them, thereby improving the utilization of existing code, saving development time, and enhancing software development efficiency [3]. Consequently, the rapid and accurate search for existing code fragments (i.e., code search) has become a crucial area of research in software engineering.

Currently, deep learning-based code search research is mainly divided into offline training and online search [4]. In the offline training phase, a deep learning network model is employed to learn features from a large dataset, and the network model parameters are acquired through learning [5]. During feature learning, the data primarily consists of Code Description pairs, where Code represents the source code fragment and Description corresponds to the statement that describes the function of the source code fragment [6].

B. Luo et al. (Eds.): ICONIP 2023, CCIS 1965, pp. 3–14, 2024.
https://doi.org/10.1007/978-981-99-8145-8_1

The deep learning network model is primarily utilized to learn the syntax and semantic relationships between code language (Code) and natural language (Description), ultimately determining the network model parameters that can map Code and Description. In the online search stage, developers input query content (describing code fragments or representing code functions), and the query content is matched with the code fragments in the dataset to obtain the highest matching results [7]. While Code Description pairs are employed for feature learning in offline training, Query instead of Description is used to match Code in online search. However, there are two differences between Query and Description. Firstly, there is a difference in length, with Query usually being shorter than Description [8]. According to statistics, the average query length entered by search personnel is 2–3, while the average description length for code is 20–30 [9]. Secondly, there are semantic differences between Query and Description. Description refers to the way code developers describe code fragments from their perspective, while Query represents the description of code fragments based on the needs of search personnel, from their own perspective. These differences lead to significant differences in the descriptions of the same code fragment by developers and searchers [10]. Nevertheless, existing research often treats Query and Description equally, ignoring their differences, which can have a significant impact on search results [11].

To address the differences between Query and Description in existing code search research, researchers have proposed query extension studies. Query extension research aims to design extension methods and sources to expand Query, reduce the differences between Query and Description, and improve the accuracy of code search results [12]. Existing research on query extension primarily focuses on two aspects: extension methods and extension sources [13]. Extension methods are primarily divided into keyword extension and source code API extension methods [14]. Keyword extension involves using words in a query statement as keywords, matching them with words in a vocabulary, and extending the query statement with words that have high similarity [15]. The source code API extension method involves using APIs in code fragments as extension sources, matching query statement keywords with API keywords, and extending query statements with API keywords that have high similarity [16]. Existing research on extension sources mainly relies on Q&A pairs (Stack Overflow Q&A) and WordNet as extension sources in question answering libraries [17]. However, existing extension methods and sources for query extension research are still unable to effectively reduce the differences between Query and Description, and cannot effectively improve the query extension effect, resulting in lower code search results. Overall, there are still three main issues with existing research on query extension. (1) The current datasets available for research in code search primarily focus on code search itself, and there is a lack of specialized datasets for query extension research. (2) Disregarding the differences between Query and Description makes it challenging to map Query and Description effectively. (3) The lack of consideration for query intent results in an inability to accurately express search intent.

In order to enhance the effectiveness of query expansion, we propose a multi-intent description of keyword expansion model for code search, abbreviated as MDKE-CS. Firstly, we construct a Query Code Description keyword dataset from the perspectives of code developers, searchers, and reviewers based on the Q&A of Stack Overflow.

Secondly, we utilize a deep learning model to train Query Code data pairs to obtain the best matching Code for Query. Then, we use multiple types of Keywords corresponding to the matched Code and residual information Description to compensate for keyword features and extend the Query, obtaining the first extended Query$_{\text{-1}}$. We repeat the best extension times for the extended model to obtain the final Query$_{\text{-n}}$. Finally, we match Query$_{\text{-n}}$ with the Code in the database one by one to obtain the best code fragment. Our proposed model, MDKE-CS, provides an approach to improve the accuracy of code search results by accounting for the differences between Query and Description while considering multiple intent descriptions for keyword expansion. By leveraging the deep learning model and multi-intent keyword expansion strategy, MDKE-CS enables effective query expansion and improves the accuracy of code search results.

In this research, we make the following contributions:

a. We create a Query-Code-Description-Keyword dataset for query extension research and demonstrate its effectiveness through experimental analysis based on existing code search models.
b. We propose a query extension code search model, MDKE-CS, which utilizes multiple intention description keywords to improve the accuracy of code search results.
c. We conduct a comparative experimental analysis on the code search performance of MDKE-CS based on the Query-Code-Description-Keyword dataset and verify the effectiveness of the proposed model in improving the accuracy of code search results. Our research provides a novel approach to enhancing the accuracy of code search by utilizing a multi-intent keyword expansion strategy and deep learning techniques in a query extension model.

The remaining sections of the paper are organized as follows: Sect. 2 introduces our proposed MDKE-CS model for query extension in code search. Section 3 presents a detailed analysis of the experimental results, including experimental preparation, dataset construction, and analysis of the experimental results. Section 4 provides a summary of our work and outlines the contributions of our research. By organizing the paper in this manner, we aim to provide a clear and structured presentation of our proposed model and experimental results, and to provide readers with a comprehensive understanding of our research.

2 Model Method

In order to make up for the shortcomings of existing query extensions, we propose a query extension code search model MDKE-CS based on multiple intention description keywords. The overall framework of the MDKE-CS model is shown in Fig. 1.

The MDKE-CS model consists of two main parts: offline training and online search. In the offline training section, the primary objective is to learn the Query-Code mapping relationship and obtain the parameters of the feature extraction model. Conventional code search training methods are utilized, and the Query-Code dataset is used for model training. A deep learning model is utilized to extract and learn the feature information of Query and Code, and similarity calculation is employed for matching analysis. The loss function is utilized as the basis for adjusting model parameters. By adjusting the

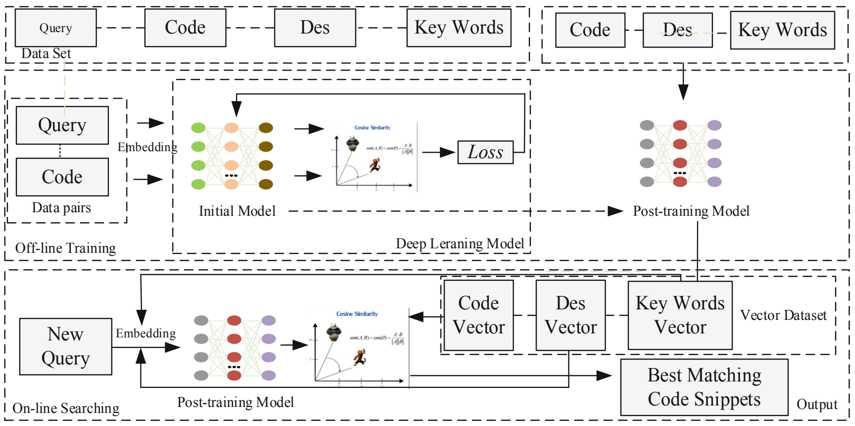

Fig. 1. The Framework of MDKE-CS

parameters of the model through a large amount of Query-Code data, the final model parameters are obtained. Finally, the Code-Description Keyword data pairs in the dataset are embedded using the model to obtain the Code-Description Keyword vector dataset pairs. The online search part primarily involves expanding the query through the expansion method to obtain the expanded query and taking the expanded query statement as the search to obtain the best search results. In the online search, the new query statement is first vectorized using the trained model, and then matched with Code to obtain the best Description vector and Keyword vector. The Description vector and Keyword vector are then extended to the query statement. The search and expansion process is repeated until the best results are achieved.

2.1 Training Model Selection

Although our research focuses on code search, our focus is on query extension methods, without studying the heterogeneous representation models between code language and natural language. Therefore, during the research process, we adopted the Bidirectional Long Short Memory Network (BiLSTM) (as shown in Fig. 2) as a deep learning model for heterogeneous feature extraction. The reason for choosing the Bidirectional Long Short Memory Network (BiLSTM) is because it is based on the manuscript "Deep Code Search" studied by Gu et al. and proposes a DeepCS code search model. The proposal of the DeepCS model represents the beginning of the introduction of deep learning into code search research, aimed at bridging the semantic gap between code language and natural language. Moreover, the DeepCS model has been recognized by a large number of researchers, and research on code search based on "Deep Code Search" and deep learning has developed rapidly.

Figure 2 depicts the structure of the BiLSTM model, which consists of three layers: the Data layer, the LSTM extraction layer, and the feature information hiding layer (h). The Data layer contains n units of data, denoted as Data_n. The LSTM extraction layer is composed of a forward LSTM and a reverse LSTM. The forward $\text{LSTM}_{1\text{-}n}$ is determined not only by the current input data Data_n, but also by the preceding $\text{LSTM}_{1\text{-}(n\text{-}1)}$ output. In

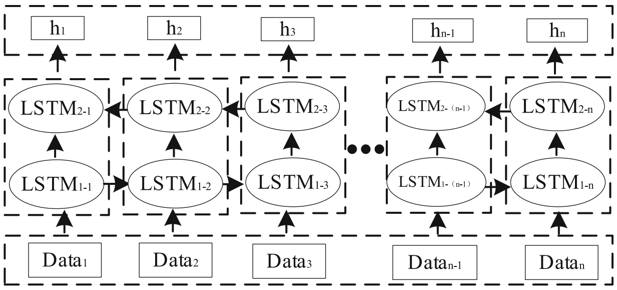

Fig. 2. Structure of the BiLSTM model

contrast, the reverse LSTM$_{2\text{-}(n\text{-}1)}$ output is influenced not only by the preceding output LSTM$_{2\text{-}n}$, but also by the output of the forward LSTM$_{1\text{-}(n\text{-}1)}$. The hidden layer hn is jointly obtained by LSTM$_{1\text{-}n}$ and LSTM$_{2\text{-}n}$ and serves as the output layer, representing the feature information of the input data Data$_n$.

2.2 Joint Embedding

As source code belongs to programming languages and query statements belong to natural languages, there exists a significant semantic gap between the two. Joint embedding is a technique that utilizes a deep learning model to embed source code and query statements into the same vector space. By embedding the two types of data in the same vector space, cosine similarity can be used to calculate the similarity between them, thereby reducing the semantic gap between the two. This technique is commonly employed in code search models to improve the accuracy of search results by accounting for both the programming language and natural language aspects of the query.

As previously mentioned, we constructed a Query-Code-Description-Keyword dataset suitable for code search query extension research. During offline training, our training objectives differ from those of "Deep Code Search." In our training, our goal is to train the mapping relationship between Query and Code in the quad metadata set. Therefore, we replaced the Description in "Deep Code Search" with Query. The Description and Keyword in the quad metadata set serve as query extension data. In our study, sequence preprocessing was performed on the source code to obtain $M = \{m_1,m_2,m_3...m_M\}$, $A = \{a_1,a_2,a_3...a_A\}$, $T = \{t_1,t_2,t_3...t_T\}$, $Q = \{t_1,t_2,t_3...t_Q\}$, respectively. The Methodname sequence contains M words, the API sequence contains A words, the Token sequence contains T words, and the Query sequence contains Q words. To better illustrate joint embedding, we will use $A = \{a_1,a_2,a_3...a_A\}$ as an example. Using BiLSTM for feature extraction and learning of the A sequence, the API sequence corresponds to the Data layer in Fig. 3. If the API sequence contains a total of A words, it corresponds to n Data values in Fig. 3. The LSTM layer performs feature extraction on the API sequence to obtain the output layer feature vector values (as shown in formula 1).

$$\{h_1, h_2, h_3...h_A\} = \mathrm{BiLSTM}(\{a_1, a_2, a_3...a_A\}) \tag{1}$$

To effectively characterize data features and reduce the impact of noise, we utilize a maximum pooling network to select the extracted features. The maximum pooling network is used to select the optimal hidden layer and obtain feature information. The maximum pooling calculation is shown in formula 2.

$$\mathrm{a} = \max pooling([h_1, h_2, h_3, ...h_A]) \tag{2}$$

After performing the maximum pooling calculation, the final feature information h_{at} is obtained (as shown in formula 3).

$$h_{at} = \tanh(W^M [h_{t-1}; a_{tA}]) \tag{3}$$

where, W^M is the parameter matrix of the API sequence in the BiLSTM network, and a_{tA} is the vector corresponding to the words in the sequence. After feature extraction in the LSTM layer, the output sequence of the hidden layer of the model is $h = \{h_1, h_2, h_3, \ldots, h_A\}$.

Similarly, we utilize BiLSTM to extract features from the other three sequences, obtaining the feature hiding layer vectors of the Methodname sequence vector (m), the Tokens sequence vector (t), and the Query vector (q), respectively. Assuming the number of words in the Methodname sequence is M, the number of words in the Token sequence is T, and the number of words in the Query sequence is Q (as shown in formula 4).

$$\begin{aligned} \mathrm{m} &= \max pooling([h_1, h_2, h_3, ...h_m]) \\ t &= \max pooling([h_1, h_2, h_3, ...h_T]) \\ q &= \max pooling([h_1, h_2, h_3, ...h_Q]) \end{aligned} \tag{4}$$

During the model training process, our goal is to train the mapping relationship between Code and Query. To achieve this, we use the concatenation network "concat" to concatenate vectors a, m, and t to obtain the source code vector c).

2.3 Extended Research

To address the limitations of existing query expansion methods, we propose a multi-intent description keyword expansion model. Our approach integrates the intention description keywords of developers, searchers, and reviewers to improve the accuracy of extension and reduce the semantic gap between queries and descriptions. The multi-intent description keyword expansion method is illustrated in Fig. 3.

Figure 3 illustrates the three-step process of the multi-intent description keyword expansion method. First, a similarity matching is performed with the code database to obtain n sorted best matching codes. Second, from the n best matching codes, the first k keywords corresponding to the code are selected as the extension words for the Query. To compensate for the loss of contextual semantics during the extraction of multi-intent description keywords, we use the Description as residual to supplement the lost semantic information. Finally, the multi-intent description keyword expansion is repeated on the Query until the decision maker meets the set requirements, and outputs the final code sorting result.

The multi-intent description keyword expansion method offers three advantages for code search tasks. First, we utilize the Stack Overflow Q&A dataset, which contains real

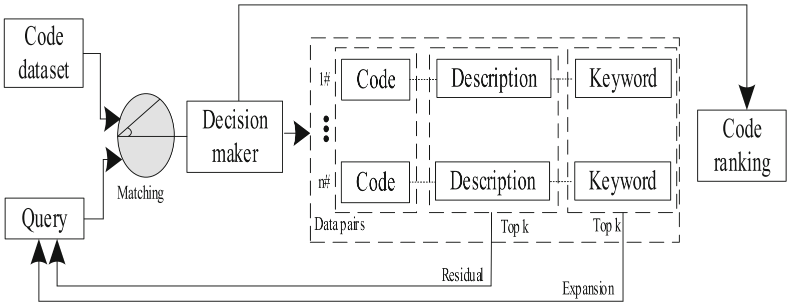

Fig. 3. Extension method

Queries and Descriptions. Queries are proposed by searchers and express their requirements, while Descriptions are provided by code developers and third-party researchers, expressing their descriptions of the requirements. Hence, our data selection is more aligned with real-world code search tasks. Second, we use extended kezywords derived from Descriptions, and use Descriptions as residuals to compensate for any lost information. As the extended information comes from the code, it can match the Query more accurately during the search. Third, we adopt a multi-intent fusion extension method that better considers the intentions of searchers, developers, and other researchers, thereby improving the accuracy of code search.

3 Experimental Analysis

To evaluate the effectiveness of the proposed model in code search tasks, we conducted a comparative experimental analysis on a Linux server. The experimental analysis includes three parts: description and query difference analysis, extension module effect analysis, and extension comparison analysis. Due to the large size of our model parameters and experimental dataset, a separate CPU cannot calculate our model parameters separately. Therefore, we used a Linux server with two Nvidia GTX 2080Ti GPUs, each with 11GB of memory. For the experiments, we implemented the proposed model using Python (version 3.6+) and the PyTorch (version 0.4) experimental simulation platform.

3.1 Experimental Preparation

To address the lack of suitable datasets in existing query extension research, we constructed a four-metadata set [Query-Code-Description-Keyword] based on the Q&A library of the Stack Overflow platform. We defined this dataset as CSExpansion. Query is a question raised by researchers on the Stack Overflow platform regarding the required code. The Description includes not only the explanation of the code developer, but also the explanation of other researchers who participated in the discussion of the code. Keywords are extracted from the Description, using the top 10 words of TFIDF in each Description. To account for the varying length of each Description, we set less than 10 keywords and fill them with 0. If there are more than 10 keywords, we sort the top 10 keywords and delete the remaining ones. Due to the limited data resources and model characteristics of the Stack Overflow platform, our CSExpansion dataset currently only includes Python and Java languages. Table 1 shows the composition of the CSExpansion dataset.

Table 1. CSExpansion Dataset

Data	Python	Java	Total
Number	37234	31297	68531

During the experimental analysis, we used four basic models: DeepCS, UNIF, WordNet, and BM25. DeepCS and UNIF models were used as the basic code search models for feature extraction and data training. WordNet and BM25 models were used as query extension models for comparative analysis. The four basic models used in our experimental analysis are DeepCS [10], UNIF [18], WordNet [19] extension model, and BM25 [20] extended model.

Our research on the proposed model is based on the basic code search models, DeepCS and UNIF. To analyze the effectiveness of the proposed model in code search tasks, we evaluated the search performance of the model using $R@k$ ($k = 1$, $k = 5$, and $k = 10$) and MRR metrics. These metrics are commonly used in information retrieval to evaluate the accuracy of ranking algorithms. $R@k$ measures the percentage of correct results in the top k returned results, while MRR measures the average rank of the first correct result. By using these metrics, we can quantitatively evaluate the effectiveness of the proposed model in improving the accuracy of code search.

3.2 Difference Analysis

The study of query extension focuses on reducing the differences between Query and Description in the code search process, thereby improving the accuracy of code search. To analyze these differences, we used the CSExpansion dataset and conducted a comparative experimental analysis using three metadata pairs (Query, Code, Description). We compared and analyzed the Query-Code and Code-Description data pairs in the experimental study. To evaluate the effectiveness of code search, we used the DeepCS and UNIF basic code search models for comparative analysis. We divided the CSExpansion dataset into training, validation, and testing sets in a 6:3:1 ratio, and conducted the experimental analysis. The comparative experimental results are shown in Table 2.

The experimental results in Table 2 show that using Description as the search object has better search performance than Query for the DeepCS and UNIF models on the CSExpansion dataset. For the Python language, using Description instead of Query in the DeepCS model resulted in an increase of 61.16%, 32.31%, 12.23%, and 12.12% for R@1, R@5, R@10, and MRR, respectively, while in the UNIF model, the increase was 15.07%, 8.21%, 12.67%, and 6.08%, respectively. For the Java language, using Description instead of Query in the DeepCS model resulted in an increase of 81.65%, 15.27%, 17.53%, and 71.89% for R@1, R@5, R@10, and MRR, respectively, while in the UNIF model, the increase was 23.44%, 17.05%, 36.52%, and 60.70%, respectively.

Table 2. Difference Analysis Results

Model	Input	Python				Java			
		R@1	R@5	R@10	MRR	R@1	R@5	R@10	MRR
DeepCS	Description	0.195	0.258	0.367	0.222	0.198	0.234	0.342	0.318
	Query	0.121	0.195	0.327	0.198	0.109	0.203	0.291	0.185
UNIF	Description	0.084	0.224	0.329	0.175	0.237	0.453	0.572	0.413
	Query	0.073	0.207	0.292	0.148	0.192	0.387	0.419	0.257

From the experimental results, we can conclude that there are significant differences between Description and Query in code search tasks, and using Description instead of Query yields better code search results. Moreover, the Java language showed a more significant improvement when using Description, possibly due to its widespread use in engineering and the involvement of more researchers in discussions, leading to more accurate Descriptions.

3.3 Comparative Experimental Analysis

We used the CSExpansion dataset and the widely used WordNet and BM25 query extension models to verify the effectiveness of our proposed MDKE-CS in code search tasks. While there are various existing query extension methods, fair comparative analysis is currently not possible due to the lack of publicly available source code from the authors. The experimental results are shown in Table 3.

Table 3. Comparative Analysis Results

Model	Input	Python				Java			
		R@1	R@5	R@10	MRR	R@1	R@5	R@10	MRR
DeepCS	WordNet	0.081	0.176	0.280	0.127	0.117	0.209	0.384	0.234
	BM25	0.088	0.181	0.288	0.137	0.180	0.339	0.405	0.254
	MDKE-CS	0.351	0.501	0.687	0.301	0.357	0.511	0.698	0.477
UNIF	WordNet	0.120	0.198	0.245	0.148	0.210	0.401	0.511	0.319
	BM25	0.147	0.178	0.225	0.158	0.221	0.429	0.577	0.380
	MDKE-CS	0.178	0.241	0.539	0.297	0.405	0.725	0.843	0.549

The experimental results in Table 3 show that the proposed MDKE-CS has better search performance for the DeepCS and UNIF models compared to the comparative models WordNet and BM25 on the CSExpansion dataset. For the Python language, using MDKE-CS instead of WordNet and BM25 in the DeepCS model resulted in an increase of 333.33% and 298.86%, 184.66% and 176.80%, 145.36% and 138.54%, 137.00% and 119.71% for R@1, R@5, R@10, and MRR, respectively, while in the UNIF model, the increase was 48.33% and 21.09%, 21.72% and 35.39%, 120.00% and 139.56%, 100.68% and 87.97%, respectively. For the Java language, using MDKE-CS instead of WordNet and BM25 in the DeepCS model resulted in an increase of 205.13% and 98.33%, 104.14% and 50.74%, 81.77% and 72.35%, 103.85% and 87.80% for R@1, R@5, R@10, and MRR, respectively, while in the UNIF model, the increase was 92.86% and 83.26%, 80.80% and 69.00%, 64.97% and 46.10%, 72.10% and 44.47%, respectively. The experimental results indicate that MDKE-CS has a more significant effect in the Python language, possibly due to better training and the suitability of the extended model for the language, improving the accuracy of description. Moreover, the proposed MDKE-CS outperforms the comparative models, WordNet and BM25, in code search tasks.

3.4 Analysis of Ablation Experiments

To analyze the structural rationality of the MDKE-CS model, we conducted a separate analysis of the roles of each module in the model. Specifically, we focused on the Keyword extension and Description residual effects in the MDKE-CS model. We compared four models: using Query search alone, using Query+Keyword search, using Query+Description, and using MDKE-CS (Query+Description+Keyword), based on the CSExpansion dataset. The experimental results are shown in Table 4.

The experimental results in Table 4 lead to two conclusions. Firstly, using Description or Keyword for extension results in better search performance than using Query alone, indicating that extending the Query can improve the accuracy of code search. Secondly, using Keyword for extension has a better effect than using Description for extension. Additionally, using both Description and Keyword for extension (MDKE-CS) yields the best results, indicating that the proposed MDKE-CS model can effectively improve the accuracy of code search.

Table 4. Model Structure Analysis

Model	Input	Python				Java			
		R@1	R@5	R@10	MRR	R@1	R@5	R@10	MRR
DeepCS	Query	0.121	0.195	0.327	0.198	0.109	0.203	0.291	0.185
	Query+Keyword	0.317	0.487	0.601	0.287	0.337	0.498	0.684	0.416
	Query+Description	0.297	0.417	0.579	0.259	0.309	0.457	0.611	0.409
	MDKE-CS	0.351	0.501	0.687	0.301	0.357	0.511	0.698	0.477
UNIF	Query	0.073	0.207	0.292	0.148	0.192	0.387	0.419	0.257
	Query+Keyword	0.161	0.379	0.478	0.271	0.350	0.668	0.798	0.495
	Query+Description	0.124	0.324	0.429	0.225	0.347	0.643	0.772	0.483
	MDKE-CS	0.178	0.241	0.539	0.297	0.405	0.725	0.843	0.549

4 Conclusion

In this research, we focused on code search query extension and proposed an MDKE-CS code search model. Through experimental analysis, we have shown that the proposed MDKE-CS model effectively improves the accuracy of code search. Based on our research, we have drawn the following conclusions:

a. There are significant differences between Query and Description during the code search process.
b. The CSExpansion dataset, which we constructed, is suitable for code search research and can improve the accuracy of query expansion.
c. The use of multiple intent keywords and residual descriptions to extend Query can effectively reduce the differences between Description and Query, and improve the accuracy of code search.

References

1. Di Grazia, L., Pradel, M.: Code search: a survey of techniques for finding code. ACM Comput. Surv. **55**(11), 1–31 (2023)
2. Liu, S., Xie, X., Siow, J., et al.: GraphSearchNet: enhancing gnns via capturing global dependencies for semantic code search. IEEE Trans. Software Eng. (2023)
3. Zeng, C., Yu, Y., Li, S., et al.: Degraphcs: embedding variable-based flow graph for neural code search. ACM Trans. Software Eng. Methodol. **32**(2), 1–27 (2023)
4. Zhong, H., Wang, X.: An empirical study on API usages from code search engine and local library. Empir. Softw. Eng. **28**(3), 63 (2023)
5. Hu, F., Wang, Y., Du, L., et al.: 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 (2023)
6. Li, X., Zhang, Y., Leung, J., et al.: EDAssistant: supporting exploratory data analysis in computational notebooks with in situ code search and recommendation. ACM Trans. Interact. Intell. Syst. **13**(1), 1–27 (2023)
7. Hu, H., Liu, J., Zhang, X., et al.: A mutual embedded self-attention network model for code search. J. Syst. Software 111591 (2023)
8. Martie, L., Hoek, A., Kwak, T.: Understanding the impact of support for iteration on code search. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 774–785 (2017)
9. Ge, X., Shepherd, D.C., Damevski, K., et al.: Design and evaluation of a multi-recommendation system for local code search. J. Vis. Lang. Comput. **39**, 1–9 (2017)
10. Yang, Y., Huang, Q.: IECS: Intent-enforced code search via extended boolean model. J. Intell. Fuzzy Syst. **33**(4), 2565–2576 (2017)
11. Karnalim, O.: Language-agnostic source code retrieval using keyword & identifier lexical pattern. Int. J. Software Eng. Comput. Syst. **4**(1), 29–47 (2018)
12. Wu, H., Yang, Y.: Code search based on alteration intent. IEEE Access **7**, 56796–56802 (2019)
13. Hu, G., Peng, M., Zhang, Y., et al.: Unsupervised software repositories mining and its application to code search. Software: Pract. Exper. **50**(3), 299–322 (2020)
14. Kim, K., Kim, D., Bissyandé, T.F., et al.: FaCoY: a code-to-code search engine. In: Proceedings of the 40th International Conference on Software Engineering, pp. 946–957 (2018)

15. Sirres, R., Bissyandé, T.F., Kim, D., et al.: Augmenting and structuring user queries to support efficient free-form code search. Empir. Softw. Eng. **23**(5), 2622–2654 (2018)
16. Rahman, M.M.: Supporting Source Code Search with Context-Aware and Semantics-Driven Query Reformulation. University of Saskatchewan (2019)
17. Yan, S., Yu, H., Chen, Y., et al.: Are the code snippets what we are searching for? a benchmark and an empirical study on code search with natural-language queries. In: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 344–354. IEEE (2020)
18. Cambronero, J., Li, H., Kim, S., et al.: 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 (2019)
19. Azad, H.K., Deepak, A.: A new approach for query expansion using Wikipedia and WordNet. Inf. Sci. **492**, 147–163 (2019)
20. Liu, J., Kim, S., Murali, V., et al.: Neural query expansion for code search. In: Proceedings of the 3rd ACM Sigplan International Workshop on Machine Learning and Programming Languages, pp. 29–37 (2019)