The Journal of Systems and Software 211 (2024) 111982

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

RRGcode: Deep hierarchical search-based code generation ☆

Qianwen Gou [a], Yunwei Dong [b], Yujiao Wu [a], Qiao Ke [c,*]

[a] School of Computer Science, Northwestern Polytechnical University, Xi'an, 710129, China
[b] School of Software, Northwestern Polytechnical University, Xi'an, 710129, China
[c] School of Mathematics and Statistics, Northwestern Polytechnical University, Xi'an, 710129, China

ARTICLE INFO

Keywords:
Hierarchical search
Re-ranking
Semantic comparison
Code generation

ABSTRACT

Retrieval-augmented code generation strengthens the generation model by using a retrieval model to select relevant code snippets from a code corpus. The synergy between retrieval and generation ensures that the generated code closely corresponds to the intended functionality. Existing methods simply feed the retrieved results to the generation model. However, if the retrieval corpus contains erroneous or sub-optimal code examples, there is a risk that the model may replicate these mistakes in the generated code. To tackle these problems, we propose RRGcode (**R**etrieval, **R**e-ranking, and **G**eneration for **code** generation), a deep hierarchical search-based code generation framework that fine-tunes initial retrieved code rankings, reducing the risk of replicating errors from the retrieval corpus and enhancing the generation of higher-quality, more reliable code. Specifically, it first retrieves relevant code candidates from a large code corpus. Then, a re-ranking model reconstructs the search space through a detailed semantic comparison between code candidates and the query, ensuring that only the most relevant and accurate candidates are considered. Finally, the re-ranked top-K codes, along with the query, serve as input for the code generation model. Extensive experiments are conducted to evaluate the effectiveness of generated code by RRGcode, demonstrating state-of-the-art performance in code generation tasks.

1. Introduction

Code generation has received significant attention due to its remarkable capacity for enhancing programmers' productivity and alleviating developers' workload (Xu et al., 2022a; Yang M. et al., 2022; Bonfanti et al., 2020; Öztürk, 2020). To harness the abundant knowledge within the code base, retrieval-augmented code generation has emerged (Hayati et al., 2018; Hashimoto et al., 2018; Parvez et al., 2021). Rather than generating from scratch, it generates programs based on retrieved human-written references, which potentially mitigates the difficulty compared to generation-based counterparts.

Numerous research efforts are dedicated to retrieval-augmented code generation. For example, Hayati et al. (2018) introduced RECODE, a sub-tree retrieval method that allows explicit referencing of existing code examples within a neural code generation model. Furthermore, Parvez et al. (2021) proposed a framework based on retrieval-augmented code and summary generation. However, a crucial challenge with these models is that the quality of the generated code strongly depends on the excellence of the retrieved results. Poor retrieval quality, including irrelevant or inaccurate code snippets, can adversely impact the performance of the generation model. How to find the most relevant code snippets remains a challenge.

Prior works (Hayati et al., 2018; Parvez et al., 2021) use either a text matching-based retrieval (e.g., BM25) or a semantic matching retrieval (e.g., a dense neural retriever) to gather initial candidate code snippets. Text matching-based retrieval is effective with keyword overlap, but poses a challenge when there is none. Especially considering that there might be subtle syntactic differences among certain code snippets, leading to entirely distinct semantics. Illustrated in Fig. 1, code-1 and code-2 calculate the product/sum of array elements, with contrasting elements highlighted in red boxes. With a cosine similarity of up to 98%, both code-1 and code-2 can be retrieved using sequence-based semantic matching. The right part of Fig. 1 displays the corresponding data flow graph, a critical aspect in capturing code semantics (Guo et al., 2021; Gou et al., 2024). It helps grasp how data is manipulated and propagated in the code, enhancing its semantic relevance. Therefore, data flow, as one factor, can enhance the accuracy of code retrieval.

In summary, the excellence of the retrieved results plays a pivotal role in enhancing the precision of the code generation model. This is

☆ Editor: Lingxiao Jiang.

* Corresponding author.
E-mail address: qiaoke@nwpu.edu.cn (Q. Ke).

https://doi.org/10.1016/j.jss.2024.111982
Received 21 March 2023; Received in revised form 11 January 2024; Accepted 20 January 2024
Available online 28 January 2024

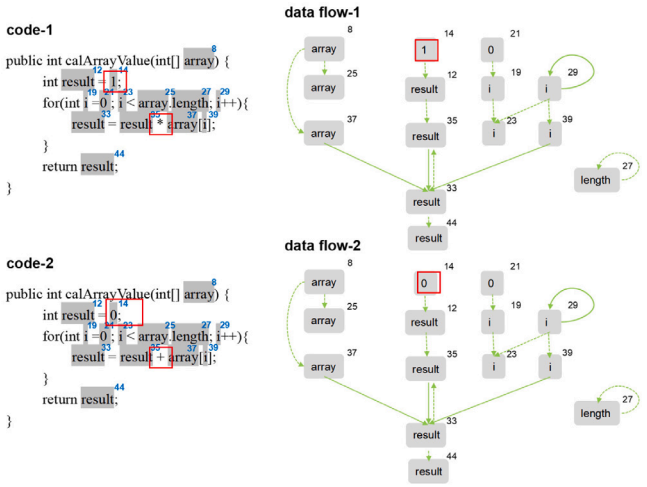


Fig. 1. An example of programs with subtle differences. Differences are highlighted with red boxes.

because incorrect retrieval results can introduce misleading information, leading the code generation model astray. The synergy between retrieval and generation ensures that the generated code closely corresponds to the intended functionality. However, existing methods suffer from the following problems:

(1) The huge search space impacts retrieval accuracy and efficiency. With massive code bases containing a multitude of snippets, modules, and functions, retrieving relevant code poses a challenge due to the vastness of the search space.
(2) Relying solely on a single retrieval method leads to constrained generation performance. Text matching-based retrieval identifies keyword matches between the query and code, while semantic matching-based retrieval may face challenges handling code snippets with diverse syntactic forms but shared semantics. Combining both approaches is essential.
(3) The lack of fine-grained semantic matching misses the crucial information between <query, code> pairs in the initial retrieval process. This can introduce errors or yield irrelevant code snippets, potentially causing the generation model to generate code that deviates from the intended functionality.

To address these problems, we introduce RRGcode, a deep hierarchical search-based method, aimed at enhancing code generation. It consists of three modules: **retrieval**, **re-ranking**, and **generation**. In the first stage, a retrieval module is used to retrieve a list of code candidates based on a query text.[1] The retrieval module uses ensemble learning to gather initial candidates (Dietterich, 2000). It integrates multiple models through a specific strategy to enhance accuracy through group decision-making. However, it may suffer from imprecise rankings and include less relevant code snippets. To address this, a re-ranking module with a cross-encoder is introduced to assess relevance more accurately. As depicted in Fig. 2, we compared the top three codes before and after re-ranking to highlight the effectiveness of the re-ranking module. After re-ranking, the order changes. This newly "top" code (C3) may provide more relevant information and the irrelevant code (C2) is removed to the back. In the third stage, the natural language query x is concatenated with the top-K codes to generate the final code. Experimental results on the CodeXGLUE dataset demonstrate that the RRGcode's superiority over other state-of-the-art models, emphasizing the importance of integrating re-ranking strategies in search-based code generation.

The main contributions of our work are as follows:

- A hybrid retrieval, merging text and semantic matching-based approaches, resulting in improved initial retrieval accuracy and relevance.
- Leveraging a cross-encoder, the re-ranking module conducts a detailed semantic comparison to fine-tune the ranking of initial retrieved codes. The fine-tuned ranking ensures the generation of higher-quality and more reliable code.
- Experiment results conducted on the CodeXGLUE dataset demonstrate that RRGcode can improve the accuracy of code generation by 10.41%, 6.03%, and 10.29% in BLEU, EM, and CodeBLEU metrics.

2. Related works

2.1. Pre-trained language model

Inspired by the enormous success of pre-training in NLP (Petroni et al., 2019; Roberts et al., 2020; Devlin et al., 2019; Vaswani et al., 2017; Radford et al., 2019), pre-trained models for programming languages also promote the development of code intelligence. Liu et al. (2019) proposed RoBERTa is a pre-trained model on text corpus with Masked Language Modeling (MLM) learning objective, while Roberta (code) is pre-trained only on code. Feng et al. (2020) proposed CodeBERT that is pre-trained on <query, code> pairs with MLM and replaced token detection learning objectives. Guo et al. (2021) proposed GraphCodeBERT, a pre-trained model that uses the semantic structure information of code to learn the feature representation of code. Guo et al. (2022) proposed a Unixcoder model, a unified cross-pattern pre-trained model for programming languages. Ahmad et al. (2021) proposed PLBART that is pre-trained on many source codes and natural language text by denoising auto-encoder. Most of the existing pre-trained models are trained on a large number of natural language query and code data, which can be widely used in software engineering-related tasks. Many code generation tools have utilized pre-trained language models. The GPT-4 pre-trained model proposed by OpenAI (2023) is a large multimodal model with strong generation capabilities and wide applicability that accepts image and text inputs and emits text outputs. It can be used to generate various types of code, including HTML, CSS, JavaScript, Python, etc. The latest ChatGPT tool utilizes the GPT-4 model, which has powerful capabilities. Copilot is a code generation tool developed in collaboration between GitHub and OpenAI. It is trained on the GPT-3 model and can automatically generate code based on natural language descriptions. RRGCode diverges from the pre-trained language model reliance on parameter-based storage, opting instead for an approach that involves explicit knowledge representation for storage purposes.

2.2. Retrieval augmented generation

Retrieval augmented generation methods are widely used in natural language text processing (Xu et al., 2022b; Zhao et al., 2022; Min et al., 2021; Guu et al., 2020; Lewis et al., 2020). RAG Model (Guu et al., 2020) and REALM model (Lewis et al., 2020) introduce retrieval into the conditional generation that is supported by retrieval in an indexed corpus of paragraphs. Using corpora as a source of knowledge, these models can extend the information available to the model by tens or even hundreds of gigabytes, significantly saving computational costs.

Retrieval-Augment code generation has been gradually applied to software engineering tasks, such as code generation and summary generation (Zhu et al., 2022a; Zhang et al., 2020; Liu et al., 2021). Hayati et al. (2018) proposed a retrieval-based approach to neural code generation, a sub-tree retrieval-based approach that explicitly references

[1] For brevity, we interchangeably use "source code", "code", and "code snippet" to denote the source code, and use "query text", "query", and "text" interchangeably to denote the natural language description of the query.

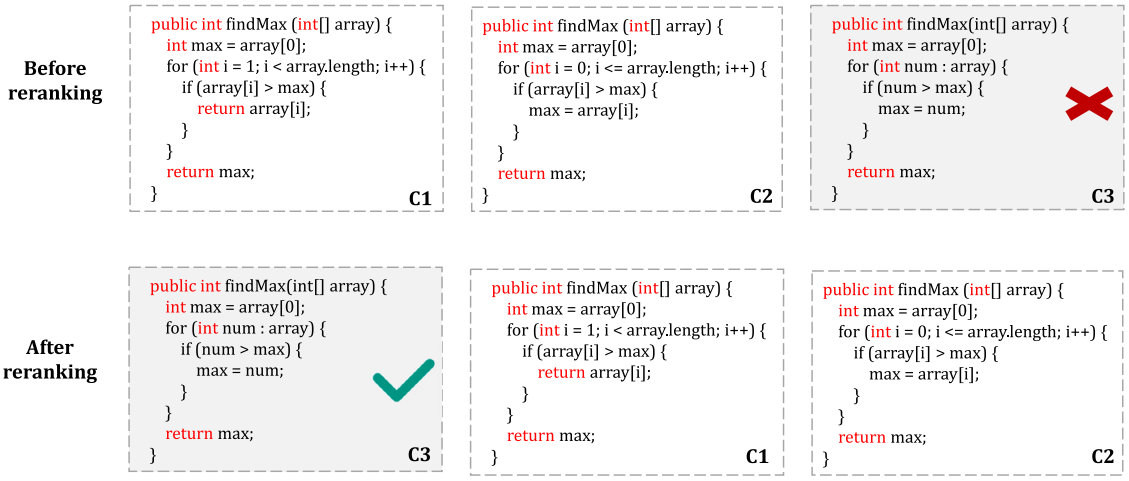


Fig. 2. An example of code comparison before and after re-ranking.

existing code examples in the neural code generation model. Hashimoto et al. (2018) pointed out in their literature that editing existing programs can be more straightforward compared to creating intricate programs from scratch. Therefore, they proposed a method to predict structured output based on retrieval and editing. The approach first retrieves training examples from input (such as natural language text) and then edits them into desired output (such as code). Parvez et al. (2021) proposed a framework based on retrieval enhancement for code generation and summary generation, which retrieves relevant codes or summaries from a corpus and uses them as a complement to the code generation or summary model. Zhu et al. (2022b) proposed a novel kNN-Transformer approach for automatic code comment generation, effectively combining nearest neighbor retrieval with a transformer-based model. Yang et al. (2022) introduced CCGIR, an information retrieval-based approach, to enhance code comment quality in smart contracts by reusing comments from similar code segments. Lu et al. (2022) introduced ReACC, a Retrieval-Augmented Code Completion framework which transforms the code completion task by using an incomplete code snippet as a query to fetch analogous code snippets from a search corpus.

In summary, retrieval-augmented generation methods can provide some relevant code snippets for the generation model to improve the accuracy of code generation. But it has the problems of the accuracy and efficiency of search and fine-grained semantic matching between code and query. However, RRGCode integrates retrieval, re-ranking, and generation modules. These modules collaborate to re-edit the retrieved programs, thereby enhancing the model's generalization capability.

3. Approach

In this section, we present the RRGcode framework, illustrated in Fig. 3. RRGcode leverages a deep hierarchical search to augment code generation. The hierarchical search process is expanded into two stages: in the first stage, a retrieval module is used to retrieve a relevant list of, e.g., 100 code candidates from a large code base to reduce the search space. Then, the second stage introduces a re-ranking model, an essential component for conducting fine-grained semantic comparisons within the narrowed search space. The meticulous semantic comparison in a narrowed search space ensures a high degree of relevance in the final selection. The framework mainly includes three modules:

(1) **Retrieval**. The retrieval module combines two retrieval methods, namely semantic matching and text matching. The part marked as ① in Fig. 3 is the training stage of the semantic retrieval model. It captures rich semantic information of natural language text and code. The part marked as ② in Fig. 3 is the stage of code embedding. The trained code encoder converts all the codes in the code base into corresponding vector representations that are then stored. For a query text, the text encoder is used to encode it to obtain the vector representation. And then, the method uses Maximum Inner Product Search (MIPS) (Karpukhin et al., 2020; Shrivastava and Li, 2014; Guo et al., 2016) to retrieve relevant codes. Besides, the BM25 algorithm (Robertson et al., 2009) is used as a text-matching retrieval method to retrieve relevant codes. Details are shown in Section 3.1.

(2) **Re-ranking**. The results retrieved by the retrieval module are merged as code candidates which are then separately input into the re-ranking model with the input text q. According to the model output, these candidates are re-ranked, from which we select top-K codes. Details are shown in Section 3.2.

(3) **Generation**. The input text q and top-K codes, as depicted in Fig. 4, are concatenated as the input for the generation model, aiming to generate the corresponding code. Further details can be found in Section 3.3.

3.1. Retrieval

The retrieval module aims to retrieve related code candidates from a code base using a natural language query. Drawing inspiration from ensemble learning, several techniques are being contemplated for assembling the retrieval module. To address the inherent complexities of code retrieval, the retrieval module adopts a dual-method approach, combining text matching and semantic matching. The integration of these two methods serves a purposeful rationale: the text matching-based method specializes in keyword matching, facilitating the identification of codes based on explicit textual similarities. Meanwhile, the semantic matching-based method focuses on capturing nuanced semantic relevance, delving into the implicit meaning embedded within the natural language query and code snippets. The retrieval module, illustrated in Fig. 5, strategically combines text matching and semantic matching. This synergy boosts the retrieval process comprehensiveness and addresses diverse code relevance facets.

3.1.1. Text matching-based retrieval

BM25 (Best Match) is a widely used ranking function in information retrieval. BM25 takes into account both the term frequency and document length to calculate a relevance score for documents in response

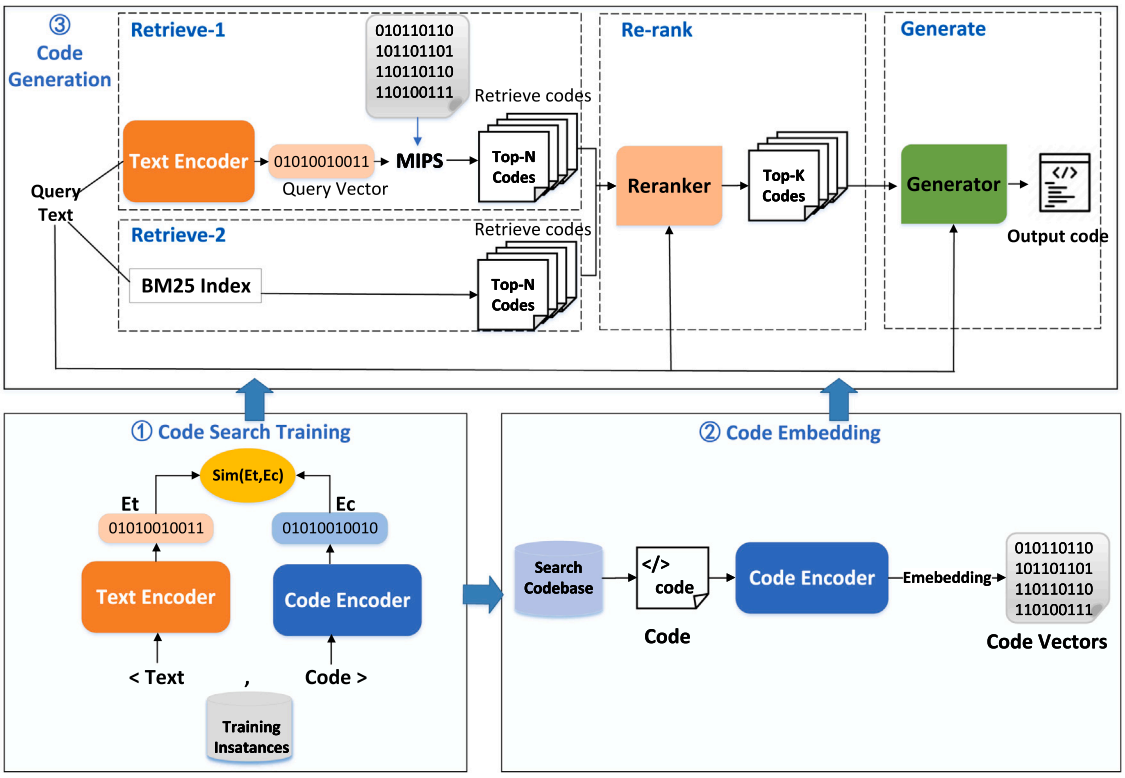


Fig. 3. The overall framework of RRGcode. ① is the training stage of semantic retrieval model. ② is the stage of code embedding. ③ is the overall code generation pipeline.

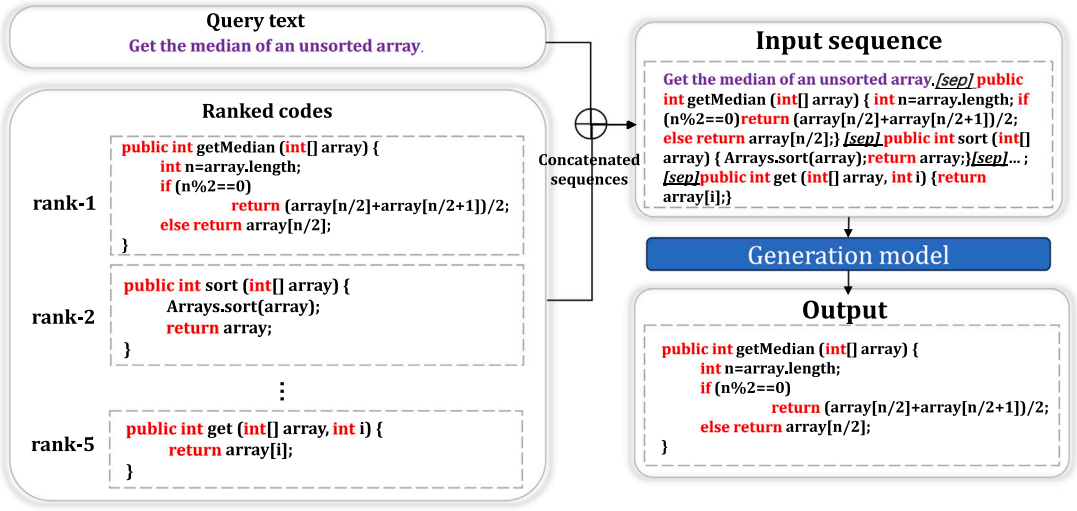


Fig. 4. An example of the input to generation model.

to a query. It has become popular due to its effectiveness in handling varying document lengths and improving search accuracy. Given a query $Q = \{q_1, \ldots, q_n\}$, the BM25 score between Q and D is calculated as follows:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avg\_doc\_len}})} \quad (1)$$

IDF (q_i) represents the inverse document frequency of token q_i. $f\ (q_i, D)$ signifies the number of times that q_i occurs in the D. $|D|$ is the length of program D in terms of tokens. avg_doc_len denotes the average program length across the corpus. The parameters k_1 and b are adjustable constants that influence the behavior of the BM25 algorithm.

The BM25 algorithm calculates the correlation between the query and code in the code base. It then selects the highest-scoring top-N codes accordingly.

3.1.2. Semantic matching-based retrieval

While the BM25 is effective for basic ranking and retrieval, it might struggle with capturing intricate semantic relationships and synonyms in the text. To overcome this limitation, it is crucial to integrate

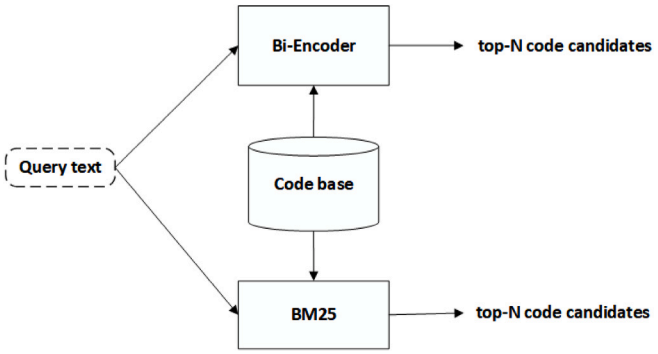


Fig. 5. The retrieval module.

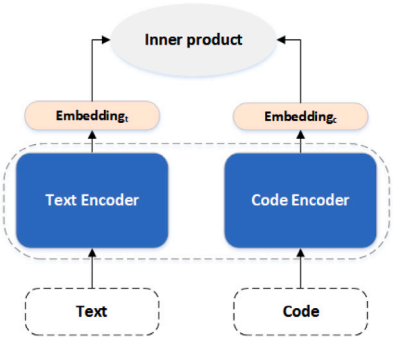


Fig. 6. The semantic retrieval model.

both keyword-based and semantic retrieval methods, enabling a more comprehensive and accurate information retrieval process.

The semantic retrieval model utilizes a bi-encoder, which comprises two encoders responsible for encoding both source code and natural language text. As shown in Fig. 6, the text-encoder and code encoder are initialized from GraphCodeBERT (Guo et al., 2021), a model that extracts the semantic structure of code by constructing its data flow graph. Our semantic retrieval model, denoted as **RRGcode-R1**(RRGcode-Retrieval) is then established. Following this, we proceed to train the model on our proprietary CodeSearchNET dataset, enhancing its performance within the specific context of code retrieval tasks.

Building upon the approach proposed by Cambronero et al. (2019), we used cosine to calculate the similarity between query and code. This entails calculating the cosine similarity between the vectors encoded by query encoder (GraphCodeBERT) Q and code encoder (GraphCodeBERT) C.

$$s(q,c) = Q(q)^T C(c) \tag{2}$$

To refine the GraphCodeBERT model, we follow the approach outlined in prior research (Balntas et al., 2016) and employ the triplet loss as the objective function. When using the q as query, we sample its matched program snippets c_+ and mismatched program snippets c_i at each mini- batch, which form positive pairs (q, c_+) and negative pairs $(q, c\_)$. Our goal is to learn a function that has a higher cosine similarity score in positive samples than in negative samples. The objective is to maximize the relevance score between the query q and its corresponding code c_+ (the positive pair (q, c_+)), while simultaneously minimizing the score between q and the non-corresponding code $c\_$ (the negative pair $(q, c\_)$). This is accomplished by:

$$L = max[s(q,c\_) - s(q,c_+) + m, 0] \tag{3}$$

where $m > 0$ is the margin parameter of the triplet loss.

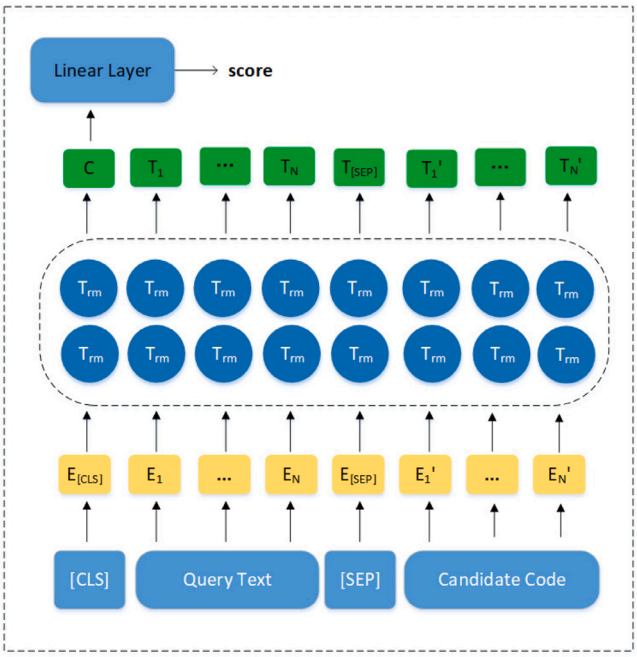


Fig. 7. The re-ranking model.

3.2. Re-ranking

The efficiency of the retrieval module is crucial, especially when dealing with vast code collections comprising millions of entries. However, there is a potential for it to yield candidates that lack relevance. Hence, the re-ranking module encompasses two critical tasks: (1) Consolidating code candidates obtained from the two retrieval methods and subsequently eliminating redundant entries. (2) Performing a fine-grained comparison between each code and query text to select related codes further.

In enhancing the final outcomes, the incorporation of a re-ranking model grounded in a cross-encoder configuration stands as a pivotal strategy. The salient attribute of cross-encoders is their heightened performance, attributed to their capacity to establish attention mechanisms across both query and code elements. Our re-ranking model, denoted as **RRGcode-R2** (RRGcode-Reranking), is initiated from the CodeBERT (Feng et al., 2020) model, renowned for its cross-encoder capabilities, as illustrated in Fig. 7. T_{rm} denotes the transformer model. T_1 through T_N represent the queries after embedding, and T_1' through T_N' represent the code segments after embedding.

Precisely, the re-ranking model simultaneously takes the query text and each code as inputs. During the computation of outputs corresponding to individual tokens within the two input sequences, the model engages in cross-attention calculations. This process facilitates the acquisition of more refined and intricate semantic associations between the two sequences.

We adopt the CodeBERT model, treating it as a semantic representation generator. [CLS] vector is a semantic feature vector representing the whole text. The output vector of the [CLS] token serves as the holistic textual representation, encompassing both the query and the code. The correlation score between code and query is obtained by feeding the [CLS] vector into a linear layer. Compared with other words in the text, this symbol, without obvious semantic information, can more fairly integrate the semantic information of each word in the text. The final codes are ranked by calculating the score of each code independently.

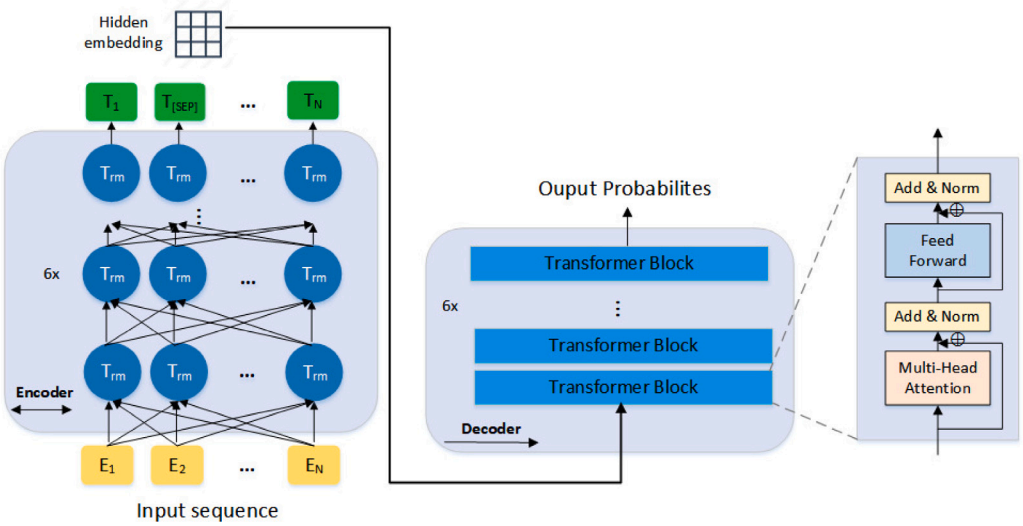


Fig. 8. The generation model.

Input. The query text q and each code c are divided into tokens. $q_1, q_2 ... q_m$ are the tokens of the query text, $c_1, c_2 ... c_n$ are the tokens of the code. [CLS], [SEP] tokens are added.

$$[CLS], q_1, q_2 ... q_m, [SEP], c_1, c_2 ... c_n \tag{4}$$

Word Embedding is used to encode and obtain the initial embedding corresponding to each token, denoted as H_0. Let the n-layer output of the Transformer be H^n,

$$H^n = transformer(H^{n-1}) \tag{5}$$

$$Q_{[CLS]} = H^n[0] \tag{6}$$

$$s = FFN(Q_{[CLS]}) \tag{7}$$

where $Q_{[CLS]}$ is the vector of [CLS], FFN is the feed-forward neural network, and s is the correlation score of query text and code c.

Similarly, the re-ranking model is also trained using the triplet loss, as the semantic retrieval model does. However, $s(q, c)$ is calculated by the re-ranking model (Eq. (7)). The loss function is the same as Eq. (3).

3.3. Generation

The goal of the generation module is to generate code with the query text and the selected top-K code candidates as input. The generation model is based on an encoder–decoder in Fig. 8. PLBART, a bidirectional auto-regressive Transformer model, is pre-trained on many source codes and natural language text by denoising auto-encoder. The code generation model is fine tuned on the PLBART pre-trained language model to generate code. The detailed model structure references this paper (Ahmad et al., 2021).

Input. We concatenate the natural language query text q with top-K related codes as the input q' of the generation model. The [sep] token is used to split the query text and code. $y_1, y_2, \ldots, y_k$ represent the top-K codes. According to PLBART, the input is truncated to a length of 512. An input example is shown in Fig. 4.

$$q' = q[sep]y_1[sep]y_2[sep]...y_k \tag{8}$$

The model is trained with the cross-entropy loss, defined as

$$L = \sum_t -log P(S_t^* \mid X, S_{<t}^*) \tag{9}$$

where S_t^* is the word at the tth position of the ground-truth output and X is input of the generation model. $S_{<t}^*$ represents the word in the first $t-1$ position of the output sequence.

Table 1
Dataset.

Dataset	Training	Validation	Testing
CodeXGLUE-Java	164 923	5183	10 955

4. Experiment

Extensive experiments are conducted to analyze the performance of RRGcode. We focus on the following Research Question (RQ):

RQ1: How effective is the RRGcode compared with the state-of-the-art approaches on code generation tasks?

RQ2: How to measure the effectiveness of the retrieval models?

RQ3: How to measure the effectiveness of the re-ranking models?

RQ4: What improvements does re-ranking bring? Can it be explained from several different aspects?

RQ5: What is the impact of the remaining parameters on the performance of RRGcode?

4.1. Dataset

To ensure the generality of the experimental results, we use the existing Java dataset in the CodeXGLUE datasets (Lu et al., 2021). We maintain the original data partitioning in the Java dataset, as shown in Table 1. The dataset contains 181 061 <query, code> pairs in total, 164 923 data for model training, 5183 data for model verification, and 10 955 data for testing.

To retrieve relevant codes, it needs to construct an extensive database to contain as much data as possible. We de-duplicated the Java datasets provided in CodeSearchNET (Husain et al., 2019) to construct the database with 1.5 million functions.

4.2. Experiment detail

The implementation of RRGcode mainly consists of three parts: retrieval, re-ranking, and generation. The three parts are trained independently.

Retrieval. The retrieval module combines two retrieval methods, namely text matching-based and semantic matching-based retrieval. We use the Anserini tool,[2] a toolkit for information retrieval, to implement

[2] https://github.com/castorini/anserini.

the BM25 algorithm. Top-100 code candidates are retrieved from the code base by using the BM25 algorithm. The semantic retrieval model is initialized from the GraphCodeBERT model, which is available from the Huggingface API. The epoch of model training is set to 15, and the batch size is set to 8. Adam is used for parameter optimization, the learning rate is set to 2×10^{-5}, linear scheduling with warm-up was used, and the dropout is set to 0.1. We also use the trained semantic retrieval model to retrieve the top 100 relevant codes from the code base.

Re-ranking. The deep learning models and heuristic algorithms are separately used in re-ranking, and the performances of the two methods are compared. The CodeBERT model is used as the classification model, and the [CLS] vector is used as the score of the code. We set the maximum length of the query sequence and code sequence to 62 and 450, respectively. The epoch of the model training is set to 10, and Adam is used to optimize the parameters. The learning rate is 10^{-5}, and l2_weight is 0.01. The heuristic algorithm uses a linear combination of the BM25 score and the score calculated by the semantic retrieval model as the new re-ranking function, defined as formula (10).

$$HF = \lambda \cdot BM25(q, c) + (1 - \lambda) \cdot sim(q, c) \tag{10}$$

Generation. Top-K codes from the re-ranked results are selected as input to the generation model, and the k value is set to 5. The generation model is initialized using the PLBART, available from the Huggingface API. The epoch of model training is set to 10, the batch size is set to 16, the beam size is set to 4, and the learning rate is set to 3×10^{-5}.

4.3. Baselines

Retrieval. In the code retrieval task, we use the following different models for comparison.

- BM25 (Robertson et al., 2009). BM25 algorithm is used to evaluate the correlation between search terms and codes. It is an algorithm based on a probabilistic retrieval model.
- RoBERTa (code) (Liu et al., 2019). RoBERTa (code) model, an improvement of Bert model, uses a larger batch size for longer training on a larger dataset.
- CodeBERT (Feng et al., 2020). CodeBERT model is a pre-trained language model that uses a mixture of masking language modeling and token detection for pre-training.

Re-ranking. We use the heuristic algorithm and deep learning models for comparison in the code re-ranking task.

- Heuristic algorithm. The semantic retrieval model uses a linear combination of BM25 scores and scores calculated as the new re-ranking function.
- RoBERTa (code). The same model was mentioned for code retrieval.
- Unixcoder (Guo et al., 2022). The Unixcoder model is a unified cross-pattern pretraining model for programming languages.
- GraphCodeBERT (Guo et al., 2021). GraphCodeBERT is pre-trained by modeling the data flow graph of the source code.

Generation. In the code generation task, we use some generative models for comparison.

- Seq2Seq. LSTM model (Hochreiter and Schmidhuber, 1997) is used as a sequence-to-sequence model.
- CodeGPT/CodeGPT-adapted (Lu et al., 2021). They are GPT-2 style models pre-trained on CodeSearchNet.
- Unixcoder. The same model mentioned for code re-ranking.
- REDCODER (Parvez et al., 2021). It is a pre-trained code generation model by providing multiple similar code snippets through a retrieval-augmented framework.

Table 2
Evaluation results of the code generation model on CodeXGLUE.

Methods	Model	Result		
		BLEU	EM	CodeBLEU
Generative methods	Seq2Seq	5.10	0.00	10.89
	CodeGPT	6.82	0.00	12.36
	CodeGPT-adapted	7.10	0.00	14.90
	Unixcoder	11.32	0.00	14.93
	PLBART	10.10	0.00	14.96
Search-based generation	REDCODER	36.72	29.37	41.50
Hierarchical search-based generation	RRGcode	**47.13**	**34.31**	**53.06**

Table 3
Evaluation results of the code retrieval model.

Methods	Result		
	MRR	R@1	R@5
RRGcode-BM25	0.0646	0.0423	0.0871
RoBERTa(code)	0.375	0.2996	0.4571
CodeBERT	0.401	0.3012	0.5134
RRGcode-R1	**0.421**	**0.3288**	**0.5255**

4.4. Experimental results and discussion

4.4.1. RRGcode vs. Baselines

For RQ1, our aim is to evaluate the effectiveness of RRGcode and measure the performance improvement it provides compared to the existing baselines. We categorize the baselines into three types: generative methods, search-based generation, and hierarchical search-based generation. Our experiments evaluate their performance using Exact Match (EM), BLEU (Papineni et al., 2002), and CodeBLEU (Ren et al., 2020) metrics on the CodeXGLUE dataset.

Results. (1) Notably, RRGcode achieves the remarkable results compared to all baselines as shown in Table 2, indicating its capability to generate more correct programs. Compared to REDCODER, RRGcode outperforms it by up to 10.41% in BLEU, 4.94% in EM, 11.50% in CodeBLEU. The significant improvements prove the superiority of RRGcode in automatic code generation. (2) RRGcode outperforms the generative methods across BLEU, EM and CodeBLEU. This is attributed to RRGcode integrating retrieval and re-ranking, enabling the extraction of relevant code from the code base. This enhances subsequent code generation, resulting in code closely aligned with desired outcomes. (3) RRGcode highlights the crucial role of integrating re-ranking strategies in search-based generation. This is attributed to the re-ranking module, which refines and optimizes the initial search-based generation results. These significant improvements prove the hierarchical search-based generation is more promising in automatic code generation.

4.4.2. Ablation study

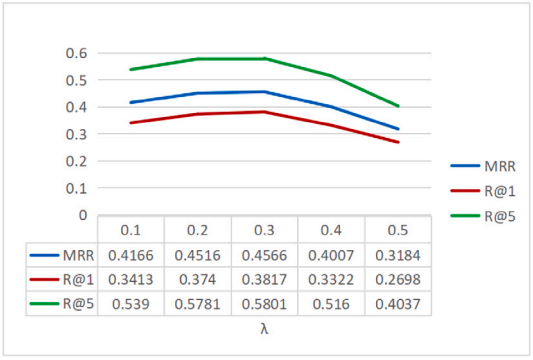
We also conduct an ablation study to evaluate the impact of different components of RRGcode.

Retrieval. RQ2 aims to assess the impact of the retrieval module and various retrieval methods on the performance of RRGcode. Table 3 presents the evaluation results, employing widely used metrics from information retrieval and code search literature: mean reciprocal rank (MRR) and Recall@K (R@K) (Ling et al., 2021). In conclusion, our experimental findings reveal the following insights: (1) RRGcode-BM25 demonstrates the lowest performance, suggesting that semantic retrieval outperforms information retrieval. (2) RRGcode-R1 exhibits the highest performance, indicating that incorporating control flow yields superior results compared to others. Moreover, this highlights the ability of control flow considerations to capture underlying, intricate semantic information within the code.

Re-ranking. RQ3 aims to assess the impact of the re-ranking module on the performance of RRGcode. We compared the impact of

Table 4
Evaluation results of the code re-ranking model.

Methods	Result		
	MRR	R@1	R@5
Heuristic_algorithm(λ = 0.3)	0.4566	0.3817	0.5801
RoBERTa(code)	0.512	0.4210	0.6925
GraphCodeBERT	0.556	0.4528	0.7069
Unixcoder	0.5544	0.4485	0.7295
RRGcode-R2	**0.569**	**0.4754**	**0.7386**



	0.1	0.2	0.3	0.4	0.5
MRR	0.4166	0.4516	0.4566	0.4007	0.3184
R@1	0.3413	0.374	0.3817	0.3322	0.2698
R@5	0.539	0.5781	0.5801	0.516	0.4037

Fig. 9. The heuristic algorithm with different λ values.

two re-ranking methods, namely the heuristic algorithm and the deep learning model, on the performance of RRGcode.

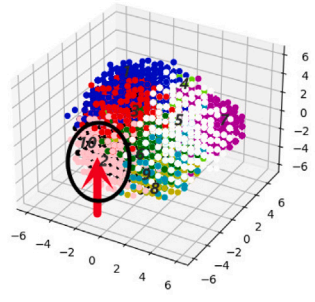
The heuristic algorithm combines the two candidate results using a weight λ. At first, we employed a heuristic algorithm, with results displayed in Fig. 9. Overall, we can find that optimal model performance is observed when λ is set to 0.3. The conclusion drawn is that after applying the heuristic algorithm for re-ranking, the performance outperforms any individual retrieval algorithm. The utilization of a linear combination to merge retrieval results might lead to a marginal enhancement in retrieval performance.

Secondly, we use deep learning models as cross-encoder for re-ranking, and the results are shown in Table 4. Results in Table 4 indicate that utilizing a cross-encoder for re-ranking generally surpasses the heuristic algorithm (λ = 0.3). The cross-encoder's strength lies in its fine-grained learning of semantic correlations between query text and code. Among these models, RRGcode-R2 works better. We attribute this improvement to the ability of CodeBERT to capture the semantic correlation between query text and code. The experimental results also prove that re-ranking can benefit the code generation method combined with retrieval.

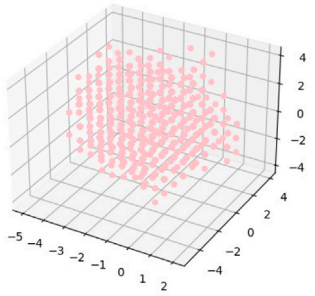
4.4.3. Visual analysis

RRGcode combines these two kinds of encoder and adopts the deep hierarchical search method. As mentioned before, the deep hierarchical search method may bring two benefits: (1) Reduce the dimension of huge search space, and reconstruct it. (2) Using a cross-encoder network for re-ranking, fine-grained semantic matching is conducted between code and text.

Reduce the dimension of huge search space and reconstruct it. In the retrieval process, a bi-encoder is first used to encode all codes in the code base. We use the K-means algorithm to cluster the code vectors in the code base and use a three-dimensional graph for visualization (Toman et al., 2020; Amory et al., 2012), as shown in Fig. 10. We set the value of k to 10. Table 5 shows the distance between different cluster centers. Let us assume that the position indicated by the red arrow in Fig. 10(a) represents the vector of the query text. The codes corresponding to the surrounding vectors (in the black curve in



(a) All the code verctors



(b) A new small space

Fig. 10. Clustering the code vectors in the code base. Various colors in Fig. 10(a) typically denote different clusters.

Table 5
The distance between cluster centers.

centers	1	2	3	4	5	6	7	8	9	10
1	0.	4.938	3.191	1.568	3.676	2.089	6.17	2.522	3.207	5.125
2	4.938	0.	3.722	4.686	4.987	3.323	6.019	2.550	1.836	4.914
3	3.191	3.722	0.	1.858	1.773	2.507	3.822	2.866	2.777	3.146
4	1.568	4.686	1.858	0.	2.424	2.220	5.001	2.734	3.177	3.964
5	3.676	4.987	1.773	2.424	0.	3.026	2.600	4.138	3.843	4.622
6	2.089	3.323	2.507	2.220	3.026	0.	5.005	1.710	1.517	5.135
7	6.170	6.019	3.822	5.001	2.600	5.005	0.	6.141	5.446	6.199
8	2.522	2.550	2.866	2.734	4.138	1.710	6.141	0.	1.182	4.425
9	3.207	1.836	2.777	3.177	3.843	1.517	5.446	1.182	0.	4.809
10	5.125	4.914	3.146	3.964	4.622	5.135	6.199	4.425	4.809	0.

Fig. 10(a)) may be selected to form a small search space, as shown in Fig. 10(b). The search space was reduced from 1.5 million to 200 codes. And then, the re-ranking model is used to re-rank these codes, which means reconstructing the code in this space.

Fine-grained semantic matching is conducted between code and text. In order to explore whether fine-grained semantic matching between code and text is conducted in the re-ranking process, we provide a Java example (code-1 in Fig. 1) to show the attention when calculating query text and code. We add the weights of 12 attention heads of multi-head attention and present the attention distribution of the last decoder layer in Fig. 11. The abscissa and the ordinate corresponding to the text sequence and code sequence. The heatmap shows the value of the cross-attention coefficient for other tokens when encoding each token. The high activation (dark red) positions with high attention weights are always focused in most encoding time. It can be seen

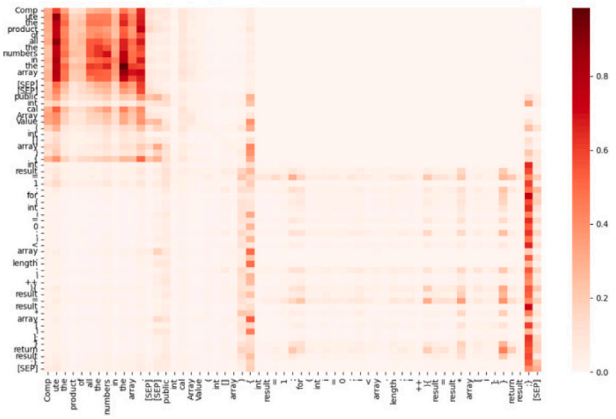


Fig. 11. Heatmap of re-ranking model for code-, where the darker colors typically represent higher attention weights.

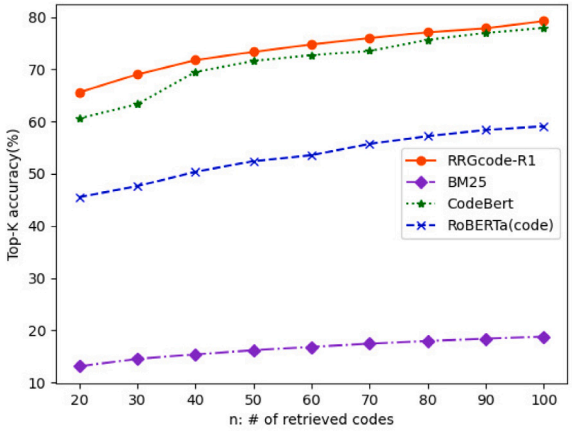


Fig. 12. Influence of the number of code candidates.

from the different colors in the heatmap that the fine-grained semantic matching between code and text is conducted.

Compare the code before and after re-ranking. In the generation step, the top five codes are taken as the input of the generation model, and there is no order requirement. We only calculate the difference rate of the codes in the top five before and after re-ranking without considering the change in order. The calculation method is as follows:

$$D = \frac{1}{N}\sum_{i=1}^{N}\frac{diff_i}{5} \tag{11}$$

where D is the difference rate, $diff_i$ is the number of different codes in the top five codes before and after re-ranking for the ith query text, and N is the number of the query text. We made statistics on the difference rate of the test set, and the result is 64%. From the result, the code changes significantly before and after re-ranking. However, according to the previous results, re-ranking also dramatically improves. The accuracy of the top five codes can be improved by more than 20%.

4.4.4. Hyper-parameters study

Influence of the number of retrieved codes. In our method, a retrieval module is used to retrieve n code candidates in a large code base to reduce the search space. We evaluate the impact of the number of code candidates on retrieval accuracy. We calculate the retrieval accuracy of n code candidates using different retrieval models, $n \in \{20, 100\}$. As can be seen from Fig. 12, the retrieval accuracy is also improving with the increase in the number of retrieved code candidates. Considering retrieval efficiency and accuracy, we set the value of n to 100.

Influence of the number of codes used for generation. Further, we evaluate the impact of different code numbers k as input on code generation, as shown in Fig. 13. As the number increases, the values of the three metrics will also increase gradually. When k is five, the growth is slow, so roughly five codes are enough to work well.

Other performance. We show the training curves of RRGcode on the CodeXGLU dataset across various training steps. Figs. 14(a) and 14(b) show the change curve of the cross entropy loss function and Negative Log Likelihood Loss (NLL-loss) during model training. As training progresses, both loss functions demonstrate a consistent downward trend, indicative of the model's learning and improvement over time. The diminishing trend and stability observed in the loss functions suggest that RRGcode is converging and approaching an optimal state. Fig. 14(c) shows the change curve of the learning rate (lr). The learning rate is dynamic and undergoes continuous adjustments throughout the training process. Smaller learning rates as training progresses allow the model to fine-tune its parameters more delicately, helping it converge to a more accurate and stable solution. Fig. 14(d) shows the memory usage (gb_free) in the training process. Monitoring memory usage is crucial, and stability in memory usage can provide insights into the efficiency of the training process. The experimental findings present the effectiveness and efficiency of RRGcode on the CodeXGLU dataset.

5. Threats to validity

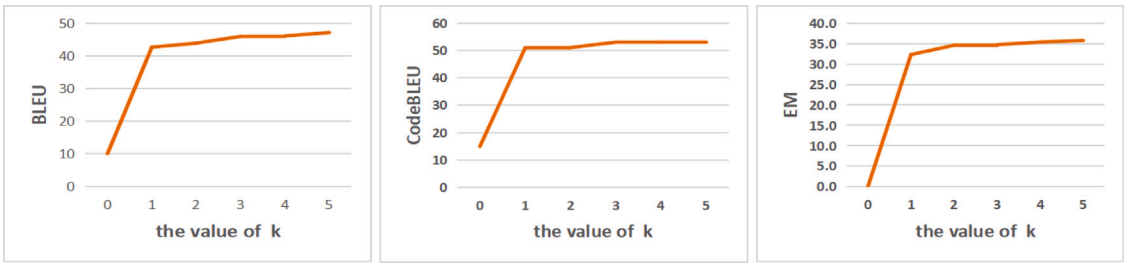
In this section, we discuss potential threats to the validity of our research findings from two perspectives: dataset noise and programming language selection.

Dataset noise. Our dataset is sourced from open platforms like GitHub, introducing the potential of noise arising from code quality, coding practices, and project diversity. This noise has the potential to hinder the effective training of our model. To mitigate this threat, we employed a filtering mechanism to exclude shorter code snippets or natural language descriptions. However, despite our best efforts, residual noise may persist within the dataset, potentially impacting the model's learning process and subsequent performance.

Programming language selection. We validated the effectiveness of our method exclusively on the Java programming language. Our belief is that the underlying methodology should be transferable to different languages. However, the intricacies of each programming language might lead to variations in performance. This limitation is inherent in any methodology that is extended across programming languages. We acknowledge the potential for differences in language syntax, semantics, and code patterns, which may affect the effectiveness of our method when applied to languages other than Java.

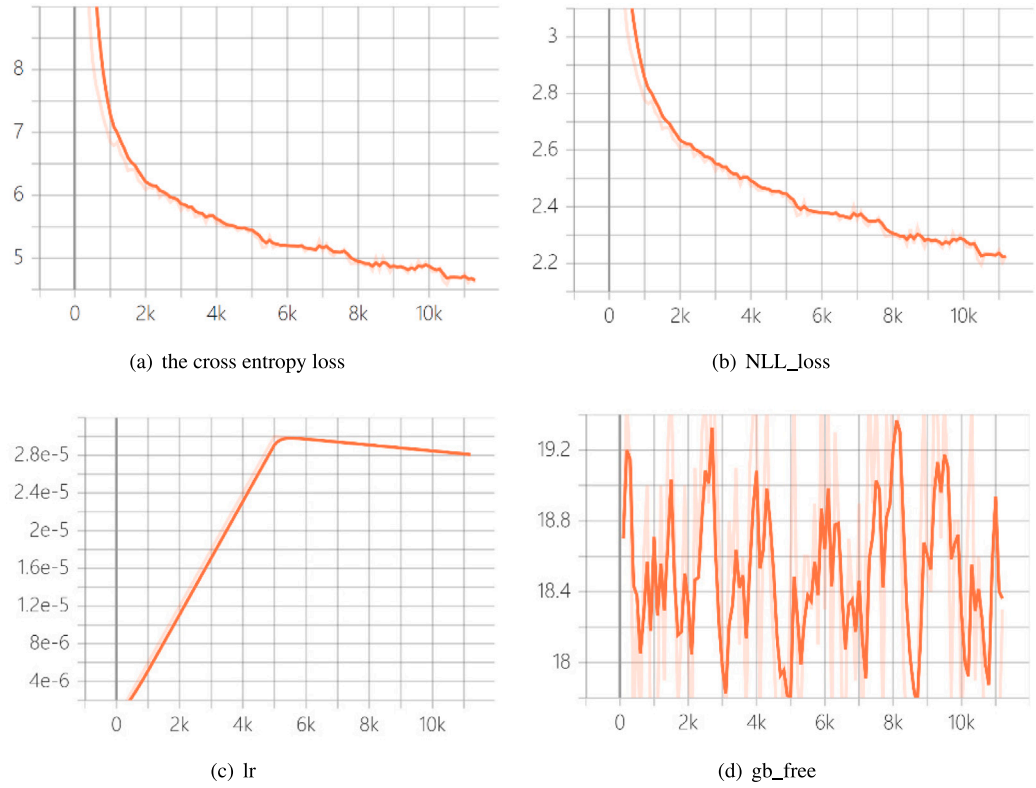
6. Conclusion and future work

We present RRGcode, a deep hierarchical search-based code generation framework that fine-tunes initial retrieved code rankings, reducing the risk of replicating errors from the retrieval corpus and enhancing the generation of higher-quality, more reliable code. We evaluated RRGcode on the CodeXGLUE dataset and showed that RRGcode could significantly improve code generation performance. However, the separate training of these three models leads to knowledge fragmentation. Therefore, future efforts should focus on their joint training to achieve a more cohesive learning outcome.



(a) BLEU (b) CodeBLEU (c) EM

Fig. 13. The impact of the number k of code candidates used for generation.



(a) the cross entropy loss (b) NLL_loss

(c) lr (d) gb_free

Fig. 14. The training curves under different steps.

CRediT authorship contribution statement

Qianwen Gou: Conceptualization, Writing – original draft, Writing – review & editing. **Yunwei Dong:** Writing – review & editing. **Yujiao Wu:** Writing – original draft, Data curation. **Qiao Ke:** Supervision, Validation.

Declaration of competing interest

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

Data availability

Data will be made available on request.

Acknowledgments

This research is supported by the Major Program of the National Natural Science Foundation of China (No. 62192733, No. 62192730), the National Natural Science Foundation of China (No. 12201498) and the Natural Science Foundation of Shaanxi Province, China (No. 2022JQ-003).

References

Ahmad, W.U., Chakraborty, S., Ray, B., Chang, K., 2021. Unified pre-training for program understanding and generation. In: Toutanova, K., Rumshisky, A., Zettlemoyer, L., Hakkani-Tür, D., Beltagy, I., Bethard, S., Cotterell, R., Chakraborty, T., Zhou, Y. (Eds.), Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021. Association for Computational Linguistics, pp. 2655–2668.

Amory, A.A., Sammouda, R., Mathkour, H., Jomaa, R.M., 2012. A content based image retrieval using K-means algorithm. In: Fong, S., Pichappan, P., Mohammed, S., Hung, P., Asghar, S. (Eds.), Seventh International Conference on Digital Information Management, ICDIM 2012, Macau, Macao, August 22-24, 2012. IEEE, pp. 221–225. http://dx.doi.org/10.1109/ICDIM.2012.6360121.

Balntas, V., Riba, E., Ponsa, D., Mikolajczyk, K., 2016. Learning local feature descriptors with triplets and shallow convolutional neural networks. In: Bmvc. p. 3.

Bonfanti, S., Gargantini, A., Mashkoor, A., 2020. Design and validation of a C++ code generator from abstract state machines specifications. J. Softw.: Evol. Process 32 (2), e2205.

Cambronero, J., Li, H., Kim, S., Sen, K., Chandra, S., 2019. When deep learning met code search. In: Dumas, M., Pfahl, D., Apel, S., Russo, A. (Eds.), Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019. ACM, pp. 964–974. http://dx.doi.org/10.1145/3338906.3340458.

Devlin, J., Chang, M., Lee, K., Toutanova, K., 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In: Burstein, J., Doran, C., Solorio, T. (Eds.), Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers). Association for Computational Linguistics, pp. 4171–4186.

Dietterich, T.G., 2000. Ensemble methods in machine learning. In: Kittler, J., Roli, F. (Eds.), Multiple Classifier Systems, First International Workshop, MCS 2000, Cagliari, Italy, June 21-23, 2000, Proceedings. In: Lecture Notes in Computer Science, vol. 1857, Springer, pp. 1–15.

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., Zhou, M., 2020. CodeBERT: A pre-trained model for programming and natural languages. In: Cohn, T., He, Y., Liu, Y. (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020. In: Findings of ACL, vol. EMNLP 2020, Association for Computational Linguistics, pp. 1536–1547.

Gou, Q., Dong, Y., Wu, Y., Ke, Q., 2024. Semantic similarity-based program retrieval: a multi-relational graph perspective. Front. Comput. Sci. 18 (3), 183209.

Guo, R., Kumar, S., Choromanski, K., Simcha, D., 2016. Quantization based fast inner product search. In: Gretton, A., Robert, C.C. (Eds.), Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016. In: JMLR Workshop and Conference Proceedings, vol. 51, JMLR.org, pp. 482–490, URL: http://proceedings.mlr.press/v51/guo16a.html.

Guo, D., Lu, S., Duan, N., Wang, Y., Zhou, M., Yin, J., 2022. UniXcoder: Unified cross-modal pre-training for code representation. In: Muresan, S., Nakov, P., Villavicencio, A. (Eds.), Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022. Association for Computational Linguistics, pp. 7212–7225.

Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., Tufano, M., Deng, S.K., Clement, C.B., Drain, D., Sundaresan, N., Yin, J., Jiang, D., Zhou, M., 2021. GraphCodeBERT: Pre-training code representations with data flow. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net.

Guu, K., Lee, K., Tung, Z., Pasupat, P., Chang, M., 2020. Retrieval augmented language model pre-training. In: International Conference on Machine Learning. PMLR, pp. 3929–3938.

Hashimoto, T.B., Guu, K., Oren, Y., Liang, P.S., 2018. A retrieve-and-edit framework for predicting structured outputs. Adv. Neural Inf. Process. Syst. 31.

Hayati, S.A., Olivier, R., Avvaru, P., Yin, P., Tomasic, A., Neubig, G., 2018. Retrieval-based neural code generation. In: Riloff, E., Chiang, D., Hockenmaier, J., Tsujii, J. (Eds.), Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018. Association for Computational Linguistics, pp. 925–930.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9 (8), 1735–1780.

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.

Karpukhin, V., Oguz, B., Min, S., Lewis, P.S.H., Wu, L., Edunov, S., Chen, D., Yih, W., 2020. Dense passage retrieval for open-domain question answering. In: Webber, B., Cohn, T., He, Y., Liu, Y. (Eds.), Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020. Association for Computational Linguistics, pp. 6769–6781.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al., 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. Adv. Neural Inf. Process. Syst. 33, 9459–9474.

Ling, X., Wu, L., Wang, S., Pan, G., Ma, T., Xu, F., Liu, A.X., Wu, C., Ji, S., 2021. Deep graph matching and searching for semantic code retrieval. ACM Trans. Knowl. Discov. Data (TKDD) 15 (5), 1–21.

Liu, S., Chen, Y., Xie, X., Siow, J.K., Liu, Y., 2021. Retrieval-augmented generation for code summarization via hybrid GNN. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V., 2019. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.

Lu, S., Duan, N., Han, H., Guo, D., Hwang, S., Svyatkovskiy, A., 2022. ReACC: A retrieval-augmented code completion framework. In: Muresan, S., Nakov, P., Villavicencio, A. (Eds.), Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022. Association for Computational Linguistics, pp. 6227–6240.

Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C.B., Drain, D., Jiang, D., Tang, D., Li, G., Zhou, L., Shou, L., Zhou, L., Tufano, M., Gong, M., Zhou, M., Duan, N., Sundaresan, N., Deng, S.K., Fu, S., Liu, S., 2021. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In: Vanschoren, J., Yeung, S. (Eds.), Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, Virtual.

Min, S., Lee, K., Chang, M., Toutanova, K., Hajishirzi, H., 2021. Joint passage ranking for diverse multi-answer retrieval. In: Moens, M., Huang, X., Specia, L., Yih, S.W. (Eds.), Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021. Association for Computational Linguistics, pp. 6997–7008.

OpenAI, 2023. GPT-4 technical report. http://dx.doi.org/10.48550/ARXIV.2303.08774, URL: https://arxiv.org/abs/2303.08774.

Öztürk, Ş., 2020. Two-stage sequential losses based automatic hash code generation using Siamese network. Avrupa Bilim Teknol. Derg. 39–46.

Papineni, K., Roukos, S., Ward, T., Zhu, W.-J., 2002. Bleu: a method for automatic evaluation of machine translation. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics. pp. 311–318.

Parvez, M.R., Ahmad, W.U., Chakraborty, S., Ray, B., Chang, K., 2021. Retrieval augmented code generation and summarization. In: Moens, M., Huang, X., Specia, L., Yih, S.W. (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021. Association for Computational Linguistics, pp. 2719–2734.

Petroni, F., Rocktäschel, T., Riedel, S., Lewis, P.S.H., Bakhtin, A., Wu, Y., Miller, A.H., 2019. Language models as knowledge bases? In: Inui, K., Jiang, J., Ng, V., Wan, X. (Eds.), Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019. Association for Computational Linguistics, pp. 2463–2473.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al., 2019. Language models are unsupervised multitask learners. OpenAI Blog 1 (8), 9.

Ren, S., Guo, D., Lu, S., Zhou, L., Liu, S., Tang, D., Sundaresan, N., Zhou, M., Blanco, A., Ma, S., 2020. Codebleu: a method for automatic evaluation of code synthesis. arXiv preprint arXiv:2009.10297.

Roberts, A., Raffel, C., Shazeer, N., 2020. How much knowledge can you pack into the parameters of a language model? In: Webber, B., Cohn, T., He, Y., Liu, Y. (Eds.), Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020. Association for Computational Linguistics, pp. 5418–5426.

Robertson, S., Zaragoza, H., et al., 2009. The probabilistic relevance framework: BM25 and beyond. Found. Trends® Inf. Retr. 3 (4), 333–389.

Shrivastava, A., Li, P., 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (Eds.), Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada. pp. 2321–2329, URL: https://proceedings.neurips.cc/paper/2014/hash/310ce61c90f3a46e340ee8257bc70e93-Abstract.html.

Toman, S.H., Abed, M.H., Toman, Z.H., 2020. Cluster-based information retrieval by using (K-means)- hierarchical parallel genetic algorithms approach. CoRR abs/2008.00150. URL: https://arxiv.org/abs/2008.00150, arXiv:2008.00150.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. Adv. Neural Inf. Process. Syst. 30.

Xu, Y., Ishii, E., Cahyawijaya, S., Liu, Z., Winata, G.I., Madotto, A., Su, D., Fung, P., 2022b. Retrieval-free knowledge-grounded dialogue response generation with adapters. In: Feng, S., Wan, H., Yuan, C., Yu, H. (Eds.), Proceedings of the Second DialDoc Workshop on Document-Grounded Dialogue and Conversational Question Answering, DialDoc@ACL 2022, Dublin, Ireland, May 26, 2022. Association for Computational Linguistics, pp. 93–107.

Xu, F.F., Vasilescu, B., Neubig, G., 2022a. In-ide code generation from natural language: Promise and challenges. ACM Trans. Softw. Eng. Methodol. (TOSEM) 31 (2), 1–47.

Yang, G., Liu, K., Chen, X., Zhou, Y., Yu, C., Lin, H., 2022. CCGIR: Information retrieval-based code comment generation method for smart contracts. Knowl.-Based Syst. 237, 107858. http://dx.doi.org/10.1016/j.knosys.2021.107858.

Yang M., G.B., Duan Z., J.Z., Zhan N., D.Y., 2022. Intelligent program synthesis framework and key scientific problems for embedded software. Chin. Space Sci. Technol. 42 (4), 1.

Zhang, J., Wang, X., Zhang, H., Sun, H., Liu, X., 2020. Retrieval-based neural source code summarization. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering. ICSE, IEEE, pp. 1385–1397.

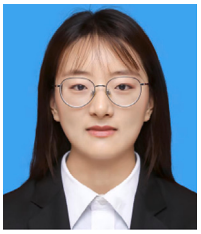
Zhao, D., Liu, X., Ning, B., Liu, C., 2022. HRG: A hybrid retrieval and generation model in multi-turn dialogue. In: International Conference on Database Systems for Advanced Applications. Springer, pp. 181–196.

Zhu, X., Sha, C., Niu, J., 2022a. A simple retrieval-based method for code comment generation. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering. SANER, IEEE, pp. 1089–1100.

Zhu, X., Sha, C., Niu, J., 2022b. A simple retrieval-based method for code comment generation. In: IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2022, Honolulu, HI, USA, March 15-18, 2022. IEEE, pp. 1089–1100. http://dx.doi.org/10.1109/SANER53432.2022.00126.

Qianwen Gou received the bachelor's degree from Northwestern University, China. She is currently a Ph.D. student with the School of Computer Science, Northwestern Polytechnical University. Her research interests include program synthesis, program recommendation.

Yunwei Dong received the doctor's degree from Northwestern University, China. His research interests include embedded systems, information-physical convergence systems, trusted software design and verification, and intelligent software engineering.



Yujiao Wu received the bachelor's degree from Henan University of Science and Technology, China. She is currently a master student with the School of Computer Science, Northwestern Polytechnical University. Her research interests include program synthesis and code search.

Qiao Ke received her Ph.D. degree in Applied Mathematics from Xi'an Jiaotong University, Xi'an, China in 2019. Her research interests are in the areas of deep learning, statistics learning, Bayesian learning, image processing, and the internet of things.