

Yet Another Combination of IR- and Neural-based Comment Generation

Yuchao Huang
Chinese Academy of Sciences
Beijing, China
yuchao2019@iscas.ac.cn

Moshi Wei
York University
Toronto, Canada
moshiwei@yorku.ca

Song Wang
York University
Toronto, Canada
wangsong@yorku.ca

Junjie Wang
Chinese Academy of Sciences
Beijing, China
junjie@iscas.ac.cn

Qing Wang
Chinese Academy of Sciences
Beijing, China
wq@iscas.ac.cn

ABSTRACT

Background: Code comment generation techniques aim to generate natural language descriptions for source code. There are two orthogonal approaches for this task, i.e., information retrieval (IR) based and neural-based methods. Recent studies have focused on combining their strengths by feeding the input code and its similar code snippets retrieved by the IR-based approach to the neural-based approach, which can enhance the neural-based approach's ability to output low-frequency words and further improve the performance.

Aim: However, despite the tremendous progress, our pilot study reveals that the current combination is not generalizable and can lead to performance degradation. In this paper, we propose a straightforward but effective approach to tackle the issue of existing combinations of these two comment generation approaches.

Method: Instead of binding IR- and neural-based approaches statically, we combine them in a dynamic manner. Specifically, given an input code snippet, we first use an IR-based technique to retrieve a similar code snippet from the corpus. Then we use a Cross-Encoder based classifier to decide the comment generation method to be used dynamically, i.e., if the retrieved similar code snippet is a true positive (i.e., is semantically similar to the input), we directly use the IR-based technique. Otherwise, we pass the input to the neural-based model to generate the comment.

Results: We evaluate our approach on a large-scale dataset of Java projects. Experiment results show that our approach can achieve 25.45 BLEU score, which improves the state-of-the-art IR-based approach, neural-based approach, and their combination by 41%, 26%, and 7%, respectively.

Conclusions: We propose a straightforward but effective dynamic combination of IR-based and neural-based comment generation, which outperforms state-of-the-art approaches by a substantial margin.

CCS CONCEPTS

• Software and its engineering → Software maintenance tools.

KEYWORDS

Comment generation, information retrieval, deep neural network

ACM Reference Format:

Yuchao Huang, Moshi Wei, Song Wang, Junjie Wang, and Qing Wang. 2018. Yet Another Combination of IR- and Neural-based Comment Generation. In *Proceedings of ACM Conference (Conference'21), June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/1122445.1122456>

1 INTRODUCTION

Manually writing comments is very time-consuming, and code comments are often low-quality, missing, or mismatched after the software is upgraded [6, 20]. To assist developers in writing high-quality comments or fill in absent comments, code comment generation techniques have been proposed, which aim to generate a summary for a given code snippet automatically [8, 16, 19, 33, 44, 49].

Most of existing code comment generation approaches can be categorized into two orthogonal types, i.e., the information retrieval (IR) based approaches [9, 12, 13, 21, 23, 28, 30, 47], which leverage the comments of retrieved similar code snippets to generate comments for code snippets and the neural-based approaches [15, 16, 19, 27], which treat the comment generation task as a translation problem and build neural machine translation (NMT) models to generate comments for code snippets. IR-based approaches can directly leverage the existing and manually written comments, which may contain rare words or project-specific information that are difficult to be generated by NMT [25]. In contrast, the neural-based approaches perform more robustly on general and new-coming samples with generalization capability [25]. Therefore, recent studies [44, 49] have gradually focused on combining the strengths of the IR-based and neural-based approaches to achieve better performance. Specifically, most of the existing approaches bind IR- and neural-based approaches statically, i.e., each input code sample and its retrieved similar code snippet from the IR-based approaches will be fed to the NMT model of neural-based approaches to generate comments regardless of whether the retrieved similar code snippet is actually similar to the input one or not. In this paper, we will refer to these approaches as *IR+NMT* approaches.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference'21, June 03–05, 2018, Woodstock, NY

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-XXXX-X/18/06...\$15.00

<https://doi.org/10.1145/1122445.1122456>

Input Sample:

```
1. public void loadProject(String arg) throws IOException {
2.     project = Project.readProject(arg);
3.     projectLoadedFromFile = true;
4. }
```

Similar sample retrieved from the corpus:

```
1. public void loadProject(String arg) throws IOException {
2.     Project newProject = Project.readProject(arg);
3.     newProject.setConfiguration(project.getConfiguration());
4.     project = newProject;
5.     projectLoadedFromFile = true;
6. }
```

Comment of the retrieved similar sample: `load given project file`

Ground Truth: `load given project file`

Figure 1: An example where the retrieved code snippet is semantically similar to the input one regarding both code and comment.

Input Sample:

```
1. public void setEnabled(boolean b) {
2.     this.enabled = b;
3.     super.setEnabled(b);
4. }
```

Similar sample retrieved from the corpus:

```
1. public void setEnabled(boolean b) {
2.     super.setEnabled(b);
3.     spinner.setEnabled(b);
4. }
```

Comment of the similar sample: `set to true if component enabled`

Ground Truth: `set the value displayer active inactive`

Figure 2: A false positive example where the retrieved code snippet is only textually, not semantically, similar to the input code sample.

However, despite the tremendous progress of existing *IR+NMT* approaches, our pilot study reveals that such a combination is not generalizable and can lead to performance degradation. For instances, Figure 1 shows an example that the comment from the retrieved similar code snippet is a perfect match to the input code sample; thus, there is no need to feed it into the neural-based models. In contrast, Figure 2 shows another example that a retrieved sample is highly lexical similar to the input sample in codes while they are irrelevant in comments; feeding the retrieved false-positive code snippets into a neural-based model will confusing the neural model and further degrade its performance.

In this paper, to tackle the issue of existing static binding of IR- and neural-based approaches, we propose a straightforward but effective approach to combine the strengths of the IR-based and neural-based approaches in a dynamic manner. Specifically, given an input code snippet, we first use an IR-based approach to retrieve a similar code snippet from the corpus. Then we use a Cross-Encoder based classifier to select the comment generation method to be used dynamically, i.e., if the retrieved similar code snippet is a true positive, we directly reuse the existing comment from the similar sample retrieved by the IR technique. Otherwise, we pass the input to the neural-based approach to generate its comment.

To evaluate our approach, we conduct experiments on a large-scale dataset provided by LeClair et al. [27], which comes from the Sourcerer repository and contains about 2M code-comment pairs. We employ BLEU [35], METEOR [2], ROUGE-L [29], and CIDER [43] as evaluation metrics to evaluate predicted comments. The experimental results show that our approach can outperform state-of-the-art baselines on all selecting metrics. Specifically, our approach can achieve 25.45 BLEU score, which improves the state-of-the-art IR-based approach, neural-based approach, and their combination by 41%, 26%, and 7%, respectively.

The main contributions of this paper are as follows:

- We propose a straightforward but effective approach to combine the IR-based and neural-based comment generation approaches in a dynamic manner.
- We have designed a Cross-Encoder based classifier, which dynamically selects the comment generation method to be used for each input sample.
- We conduct extensive experiments on a large-scale dataset to evaluate the performance of our approach. The experiment results show the effectiveness of our approach.
- We release the source code of our approach and the dataset of our experiments to help other researchers replicate and extend our study¹.

The rest of this paper is organized as follows. Section 2 presents the background of this study. Section 3 describes the details of our approach. Section 4 and Section 5 present the experiment setup and results. Section 6 discusses the strengths of our approach and threats to validity. Section 7 reviews related work. Finally, we conclude our work in Section 8.

2 BACKGROUND

2.1 Neural Machine Translation

Recent neural-based comment generation approaches [16, 17, 19, 27, 49] treat comment generation as an end-to-end neural machine translation (NMT) task and leverage the encoder-decoder Sequence-to-Sequence (Seq2Seq) model to learn the translating pattern. Specifically, at each time step t , it reads one token x_t from the input code snippet sequence $X = x_1, \dots, x_n$, then the encoder updates the current hidden state h_t :

$$h_t = f(x_t, h_{t-1}) \quad (1)$$

where f is a neural unit, e.g. GRU [4], LSTM [15].

Attention mechanism [1] is adopted to focus on the critical part of the input code during decoding. For predicting target word y_i , a context vector c_i is calculated as a weighted sum of all hidden states h_1, \dots, h_n :

$$c_i = \sum_{j=1}^n \alpha_{ij} h_j \quad (2)$$

The weight α_{ij} of each hidden state h_j is calculated as follows:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^n \exp(e_{ik})}, e_{ij} = a(s_{i-1}, h_j) \quad (3)$$

where s_{i-1} donates the last hidden state of the decoder, a is an alignment model, e.g., a Multi-Layer Perception (MLP) unit [34].

¹<https://zenodo.org/record/4757011>

At time step i , the hidden state s_i of the decoder is updated by:

$$s_i = f(y_{i-1}, s_{i-1}) \quad (4)$$

where y_{i-1} is the previous generated token. Then, the decoder generates the target sequence Y by sequentially predicting the conditional probability of a word y_i based on the hidden state s_i and the context vector c_i .

$$p(y_i | y_1, \dots, y_{i-1}, X) = g(y_{i-1}, s_i, c_i) \quad (5)$$

where g is the generator function, e.g., a MLP layer [34] along with *softmax*.

The cross-entropy loss function is used to train the Seq2Seq model, i.e., minimizing the following objective function:

$$\mathcal{L}(\theta) = - \sum_{i=1}^N \sum_{j=1}^L \log p(y_j^{(i)}) \quad (6)$$

where θ donates the trainable parameters, N is the number of training instances and L is the length of each target sequence. $y_j^{(i)}$ means the j th word in the i th instance.

2.2 Semantic Textual Similarity

To better distinguish false-positive samples, like the example shown in Figure 2, we treat *determining whether the retrieved results are similar to the input samples* as a supervised learning task. The semantic textual similarity (STS) task aims to determine the semantic similarity of a given sentence pair, which is similar to our task. The input sentence pair to the semantic classifier is the input and retrieved code snippet. The predicted label is whether the retrieved result is accurate.

Cross-Encoder [7] is one of the state-of-the-art methods for the semantic textual similarity (STS) task. The structure of the Cross-Encoder is shown in Figure 4. For the given sentence pair (s_1, s_2) , Cross-Encoder concatenates them by a special token ([SEP]) to encode them simultaneously. A multi-head attention pre-trained model (e.g., BERT [7]) is used to encode the concatenated sequence. In the encoding process, the self-attention mechanism allows two input sentences to perceive each other’s information at a fine-grained level. The embedding result is fed into a classifier layer that produces an output value \hat{y} between 0 and 1, indicating the semantic similarity.

In this paper, we use a Cross-Encoder based classifier to identify samples with accurate retrieved results. For the pre-trained model of the Cross-Encoder, we choose CodeBERT [10], which is trained on a large-scale code corpus consists of Java and five other programming languages [18]. Comparing with other pre-trained models on natural language, CodeBERT can save the effort of semantic migration from natural language to programming language during fine-tuning.

3 APPROACH

In this work, we propose a comment generation approach that combines the strengths of the IR- and neural-based comment generation approaches dynamically. The key idea of our approach is straightforward: given an input code snippet, we first use an IR-based approach to retrieve a similar code snippet from the corpus. Then

we use a Cross-Encoder based classifier to select the comment generation method to be used dynamically, i.e., if the retrieved similar code snippet is a true positive, we directly use the IR result. Otherwise, we pass the input to the neural-based approach to generate the comment. Unlike existing IR+NMT approaches [44, 49], we do not pass the information obtained by the IR-based approach to the neural network model to avoid textually similar but semantically dissimilar retrieved results to confuse the model.

3.1 Overview of Our Approach

The workflow of our approach is shown in Figure 3. Given an input sample, our approach generates its comment using the following three steps: 1) Comment generation with the IR-based technique (Section 3.2). In this step, our approach extracts the comment from the most similar sample retrieved from the corpus through the IR-based retrieval technique. 2) Evaluate the retrieved result (Section 3.3). We use a Cross-Encoder based classifier to determine whether the retrieved code snippet is similar to the input semantically. We assume that directly leveraging the existing comment from a true-positive similar sample, which may contain low-frequency words and project-specific information that hard to be generated by NMT [25, 44, 49], will be more accurate and informative than the generated result of NMT models. Therefore, when the retrieved code snippet is similar to the input, our approach will reuse the comment of the retrieved code snippet. Otherwise, we assume that the current sample needs to be inferred by generation-based methods. 3) Comment generation with the neural-based technique (Section 3.4). For the input sample whose retrieval result is determined to be inaccurate from the previous step, the neural model is used to automatically generate its comment based on the input code snippet and corresponding AST sequence.

3.2 Comment Generation with The IR-based Technique

This step aims to provide an existing comment for each input sample that may be reusable from the retrieved similar code snippet.

To identify the most similar sample for a given sample, in this work, we reuse the retrieval method of Re²Com [44], which is a code lexical similarity based retrieval method. The retrieval module of Re²Com uses the training set as the corpus. It retrieves the sample with the highest lexical similarity between code snippets based on BM25 algorithm from search engine Lucene², a widely used similarity metric. For each term in the given code snippet, its relevance score to the candidate code snippet is calculated based on the term frequency. Then, the BM25 score between the input and candidate code snippet is calculated as a weighted sum of the relevance score of each term, where the weight of each term is calculated based on its inverse document frequency. Finally, the candidate code snippet with the highest BM25 score is selected as the retrieved result. Note that, IR-based approach does not have a training process. We use the settings of BM25 from Re²Com to run our experiments.

²<https://lucene.apache.org/>

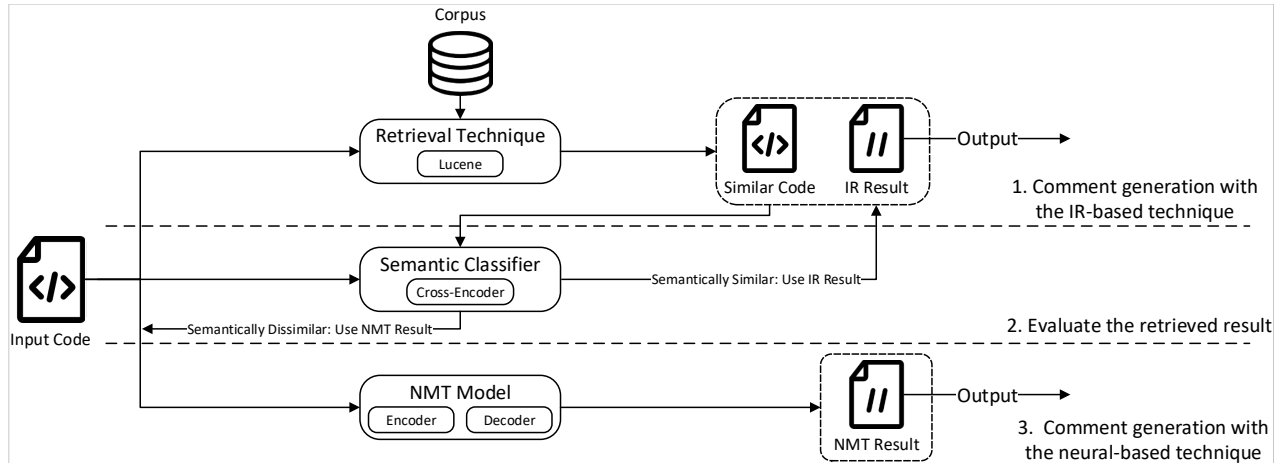


Figure 3: An overview of our approach

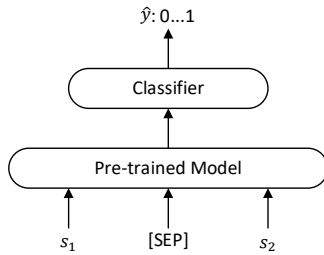


Figure 4: Structure of the Cross-Encoder

3.3 Evaluate The Retrieved Result with The Cross-Encoder based Classifier

In the previous step, we have provided an existing comment from the retrieved similar code snippet for each input sample. However, as shown in Figure 2, the results of the IR technique could be incorrect, thus to achieve more accurate determination, we compare the semantic between the input and the retrieved code snippet by a semantic model to predict whether the IR result is accurate and can be directly reused.

To identify samples with accurate IR results, we compare the input with the retrieved code snippet semantically rather than textually. This is because determining the performance of IR results from text similarity is not accurate enough. As shown in Figure 2, the input and the retrieved code snippet are very similar, with only 2-3 tokens different. However, their corresponding comments have only one token in common. In this work, we use the Cross-Encoder model for the semantic comparison, one of the state-of-the-art methods for the semantic textual similarity (STS) task. Figure 4 shows the structure of the Cross-Encoder. The input to the model is the input and retrieved code snippet. Two snippets are concatenated into a sequence through a specific token [SEP] provided by BERT [7] and simultaneously passed to a pre-trained multi-level transformer [42] network for embedding. We choose

CodeBERT [10] as the pre-trained model to save the effort of semantic migration. The embedding result of the two snippets is fed into a linear classifier layer that produces an output value between 0 and 1, indicating the degree of semantic similarity:

$$\hat{y} = T(\text{code}_{input}, \text{code}_{retrieved})W \quad (7)$$

where \hat{y} is the predicted degree of semantic similarity, W is the weight of the linear layer, and $T(\text{code}_{input}, \text{code}_{retrieved})$ is the embedding result of the input and retrieved code snippet.

The training process is fine-tuning the semantic model with pairs of code snippets to the target that if a semantically similar snippet is retrieved, the model returns 1, otherwise returns 0. We use the classic cross-entropy loss function to fine-tune the model:

$$\text{Loss} = -(y * \log(\hat{y}) + (1 - y) * \log(1 - \hat{y})) \quad (8)$$

where y indicates the golden label of whether the retrieved result is accurate.

The details of how we train the Cross-Encoder based classifier are available in Section 4.2.2.

3.4 Comment Generation with The Neural-based Technique

In the previous step, we have identified samples with accurate IR results. While the remaining input samples, we further use the neural-based approach to generate comments for them. Specifically, in this step, we first build and train an NMT model on our corpus. Then we input samples that are determined to have inaccurate IR results in the previous step to this model to generate comments. This step aims to use the generalization ability of NMT to generate comments for general input samples.

In this step, we use the state-of-the-art neural-based comment generation method, i.e., DeepCom [17]. DeepCom is an encoder-decoder structure model with the attention mechanism [1]. The input of the model contains both code and AST sequences, where the code sequence contains semantic information such as identifier names, and the AST sequence contains structural information. Using semantic and structural information from the input code snippet simultaneously can help the model understand them more clearly

and predict more accurately [17]. The model uses two encoders to encode the code sequence and the AST sequence, respectively. We follow the model training and turning processes described in DeepCom [17] to re-train the models on our corpus (details are in Section 4.2.1).

4 EXPERIMENT DESIGN

4.1 Dataset

We use the FunCom dataset provided by LeClair et al. [27] to conduct our experiments, which has been used in many existing studies [14, 27, 44]. The FunCom dataset is collected from a large Sourcerer repository [32], which contains over 50,000 projects and 5.1 million java methods. LeClair et al. treat the first sentence of the Javadoc of each method as its comment [26], use srcML [5] to extract AST sequences from source codes, then serialize them by the SBT method proposed by Hu et al. [16]. To reduce the vocabulary size, LeClair et al. adopt a series of preprocessing to the code and comment text: splitting identifiers in code and comment by camel case and underscore, removing non-alpha characters (including symbols) from the text, and converting the text to lowercase. To better simulate the case where only AST is known, identifiers in the AST sequence are replaced with <OTHER>. To reduce duplicate samples between the training and test set, LeClair et al. use a heuristic rule [39] to filter out auto-generated codes which are very similar to each other and too easy to be learned and predicted by the model. In addition, LeClair et al. divide all the data by project in the dataset building stage: data from 90% of projects are divided as the training set, 5% as the validation set, and 5% as the test set. After filtering, the FunCom dataset has about 2M code-comment pairs for training and testing.

The FunCom dataset is the most reasonable dataset to the best of our knowledge, which has a large amount of data and excludes noisy data, thus allowing us to evaluate the model’s generalization ability more accurately.

4.2 Experiment Settings

In this work, we train both DeepCom [17] and the Cross-Encoder based classifier [7] on the FunCom dataset. Their training details are as follows.

4.2.1 Training Details of DeepCom. We use the default settings of DeepCom for training, i.e., the encoder and decoder use a single-layer Gated Recurrent Unit (GRU) structure [4]. Both the word embeddings and the GRU hidden states are set to 256. In the decoding stage, beam search [46] is leveraged to obtain more accurate results, with the beam width is set to 5. We use the entire FunCom dataset for training and validation. DeepCom is trained on the FunCom training set (19,548,008 samples in total). Following DeepCom, we use Stochastic Gradient Descent (SGD) based optimizer to train the model, the initial learning rate is set to 0.5, and the learning rate decay factor is set to 0.95. In addition, to save GPU memory, we set the batch size to 256. Every 2000 training steps, the checkpoint is saved and validated on the FunCom validation set (104,273 samples in total). After 20 epochs of training (about 150,000 steps), the best parameters are selected from the checkpoint that performs best on the validation set. We trained the model on a Linux server with the

NVIDIA RTX 2060S GPU with 8GB memory, which took about 70 hours for training.

4.2.2 Training Details of Cross-Encoder Based Classifier. We use the Sentence-Bert [36] package to build and train the Cross-Encoder based classifier. In order to save the effort of language semantics migration, we adopt the widely used CodeBERT pre-trained model [10], a 24-layer bidirectional transformer [42] network.

To label the dataset for training the Cross-Encoder based classifier, we use code-comment pairs from the validation set of FunCom (104,273 samples in total). For each sample, we use the IR-based approach (details are in Section 3.2) to retrieve the most similar code snippet, and the corresponding comment will be treated as the IR result. Then we use a trained neural model (i.e., DeepCom) to generate its comment, i.e., NMT result. The label of the sample is *whether the IR result is more accurate*. Specifically, we use *sentence_bleu* metric in the NLTK [31] package to calculate the similarities of the IR result and NMT result with ground truth, respectively. If the score of the IR result is greater than the score of the NMT result, it is labeled as a positive sample; otherwise, it is labeled as a negative sample. We further exclude cases where both methods perform poorly from positive samples (e.g., both IR result and NMT result fail to hit any word in the ground truth comment). Finally, we obtain a triplet for each sample: *< Input code snippet, Retrieved code snippet, Is_IR_Result_Better? >*. After labeling the data, we take 90% of triplets (93,846 samples) for training, and the remaining 10% (10,427 samples) of triplets are used as a developmentset for tuning the parameters and testing.

We use Adam optimizer [24] to train the Cross-Encoder based classifier, and the initial training rate is set to $2e-5$, the learning rate decay factor is set to 0.99. We set the batch size to 16, and for every 2000 training steps, save the checkpoint and validate it on the development set. After fine-tuning 5 epochs (about 55,000 steps), the best parameters are selected from the checkpoint that performs best on the development set. We fine-tuned the model on a Linux server with the NVIDIA Titan RTX GPU with 24GB memory, which took about 3 hours for fine-tuning.

4.3 Baselines

4.3.1 Baselines for Evaluating Our Comment Generation Approach. To investigate the performance of our comment generation method, we selected the IR-based approach (details are in Section 3.2), four state-of-the-art neural-based comment generation methods [17, 27, 49], and two state-of-the-art IR+NMT methods [44, 49] as baselines.

1) Neural-based methods

Rencos NMT module [49] is the NMT module of Rencos [49], a standard attentional Seq2Seq model where the encoder is bidirectional LSTM and the decoder is LSTM. This baseline represents a fundamental solution to use NMT on code to comment problem, i.e., train an NMT with code as input and comment as output.

attendgru [27] is an attentional Seq2Seq-like model. This baseline predicts only one word at a time. In the encoding process, the model encodes both the code sequence and the output sequence predicted in previous steps. In the decoding process, the model predicts the next most likely word and appends it to the output sequence for the subsequent prediction steps.

ast-attndgru [27] is also an attentional Seq2Seq-like model. This baseline adds AST as an additional input to improve the prediction performance. LeClair et al. [27] use the traversal method SBT [16] to flatten the AST into a sequence and adds an additional encoder for the AST sequence.

DeepCom [17] is a standard attentional Seq2Seq model, where the encoder and the decoder are both Gated Recurrent Unit (GRU). The inputs to the model are code and AST sequences. As our proposed method takes the prediction results of this baseline as the NMT results, improvement from combining IR results can be directly measured by comparing the performance of our proposed method with this baseline.

2) IR+NMT methods

Rencos [49] combines the IR-based and neural-based comment generation by feeding the most semantic-level and syntactic-level similar code snippets of an input code snippet retrieved by IR-based approach into the neural-based approach to generate the comment. Specifically, given an input code snippet, Rencos retrieves its two most similar code snippets on semantic-level and syntactic-level. Then, the input code snippet and its two similar ones are fed separately into a trained code-to-comment NMT model to generate the comment.

Re²Com [44] uses additional encoders to encode information from the retrieved sample of IR-based approaches. For a given code snippet, a similar sample with the highest text similarity is retrieved from the corpus. Then Re²Com takes the given code, its AST, code, and comment of the similar sample as input and encodes them by four different encoders. The encoding results are fused by the similarity between the input and the retrieved code and then passed to the decoder to obtain the predicted comment.

4.3.2 Baselines for Evaluating Cross-Encoder Based Classifier. To evaluate the effectiveness of our Cross-Encoder based classifier (details are in Section 3.3) in determining whether IR results are accurate, we adopt two other classification methods as the baselines.

Lexical-level Similarity is a simple method determining whether the IR result is accurate based on the lexical similarity between the input and retrieved code. If the similarity is greater than an appropriate threshold, we assume that the IR result is accurate and treat it directly as the output; otherwise, the neural-based approach will be used to generate its comment. We follow [11] and use the *sentence_bleu* metric in the NLTK [31] package to calculate the lexical similarity. This method does not require training but needs to determine an appropriate threshold that makes the dynamic combination of IR- and neural-based approaches on the test dataset can achieve optimal performance. To find the optimal threshold, we experiment the threshold values from 0 to 1 with an interval of 0.05. When the threshold value is 0.40, this approach achieves optimal performance on FunCom’s validation set. Thus, we use 0.4 as the threshold value in our experiments.

Siamese Network [3] is another state-of-the-art method on the semantic textual similarity (STS) task. It consists of two identical encoders to encode the two input sentences separately, which share the same model structure and parameters. Then, the distance

between two embeddings is treated as the semantic similarity between the sentence pair. We use the implementation from GitHub³ to build a Siamese network, which uses a bidirectional LSTM (Bi-LSTM) [38] with 256 hidden sizes as the encoder structure and chooses manhattan distance as the similarity of embedding vector of input sentence pairs. Like Cross-Encoder, we use the labeled dataset described in Section 4.2.2 to train the Siamese network.

4.4 Evaluation Metrics

4.4.1 Metrics for Evaluating Generated Comments. In our experiments, we follow Rencos [49] and evaluate the performance of different comment generation methods with four common metrics, i.e., BLEU [35], METEOR [2], ROUGE-L [29], and CIDER [43], which are widely used in machine translation [41], text summarization [37], and image captioning [48].

BLEU [35] measures the similarity between the generated comment and ground truth by the geometric mean of n -gram matching precision scores p_n . A brevity penalty BP is used to prevent very short generated sentences.

$$BLEU = BP \cdot \exp \left(\sum_{n=1}^N w_n \log p_n \right) \quad (9)$$

where w_n is the uniform weight, and N is set to 4 in our paper. We report a composite BLEU score in addition to BLEU₁ through BLEU₄ in our experiment.

METEOR [2] calculates the similarity scores by the unigram precision P and recall R , and multiplied by a penalty of language order:

$$METEOR = \left(1 - \gamma \cdot frag^\beta \right) \cdot \frac{P \cdot R}{\alpha \cdot P + (1 - \alpha) \cdot R} \quad (10)$$

where $frag$ is the fragmentation fraction. α , β , and γ are three parameters whose default values are 0.9, 3.0 and 0.5, respectively.

ROUGE-L [29] is calculated by the Longest Common Subsequence (LCS) matching F-score. Suppose the length of the target sentence (X) and the predicted sentence (Y) are m and n , respectively, and the length of the LCS between them is $LCS(X, Y)$, then:

$$P_{lcs} = \frac{LCS(X, Y)}{n}, R_{lcs} = \frac{LCS(X, Y)}{m}, F_{lcs} = \frac{(1 + \beta^2) P_{lcs} R_{lcs}}{R_{lcs} + \beta^2 P_{lcs}} \quad (11)$$

where F_{lcs} is the value of ROUGE-L, P_{lcs} and R_{lcs} denote the LCS precision and recall, respectively, and $\beta = P_{lcs}/R_{lcs}$.

CIDER [43] examines whether the prediction result has captured the critical information. Given the generated summary c_i and the ground-truth s_i , CIDER is calculated by the frequency of n -grams and TF-IDF weighting:

$$CIDER_n(c_i, s_i) = \frac{1}{M} * \sum_{j=1}^M \frac{g^n(c_i) * g^n(s_{ij})}{\|g^n(c_i)\| * \|g^n(s_{ij})\|} \quad (12)$$

$$CIDER(c_i, s_i) = \sum_{n=1}^N w_n CIDER_n(c_i, s_i)$$

where N is set to 4, $g^n(c_i)$ denotes the TF-IDF weight vector of all n -gram in sentence c_i , M represents the number of reference sentences for each sample (in our work, $M = 1$). The final result

³<https://github.com/tlatkowski/multihead-siamese-nets>

Table 1: The performance of our method compared with other comment generation baselines (the best ones are marked in bold). The percentages in parentheses indicate the relative improvement achieved by our method compared to the IR-based method and NMT-based method (DeepCom), respectively.

Type	Approach	BLEU(%)	BLEU ₁ (%)	BLEU ₂ (%)	BLEU ₃ (%)	BLEU ₄ (%)	METEOR(%)	ROUGE-L(%)	CIDER
IR-Based	Re2Com Retrieve Module	18.04	32.04	17.84	14.4	12.88	15.41	30.64	1.643
Neural-based	Rencos NMT Module	19.15	34.64	20.58	15.11	12.49	18.92	39.54	2.074
	attendgru	19.26	38.64	21.71	14.63	11.21	19.34	40.16	1.984
	ast-attendgru	19.73	39.8	22.25	14.93	11.46	19.43	39.94	1.952
	DeepCom	20.11	40.71	22.57	15.17	11.73	19.92	40.25	2.044
IR+NMT	Rencos	19.86	36.7	21.58	15.55	12.64	19.17	39.9	2.066
	Re ² Com	23.69	40.38	24.74	19.12	16.48	20.28	39.91	2.282
Our Method		25.45 (41%/26%)	43.92 (37%/7%)	27.08 (51%/19%)	20.38(41%/34%)	17.3 (34%/47%)	22.03 (42%/10%)	43.21 (41%/7%)	2.46 (49%/20%)

CIDER (c_i, s_i) is calculated by summing of the scores for different n -grams ($CIDER_n(c_i, s_i)$) with weight w_n .

4.4.2 Metrics for Evaluating Cross-Encoder Based Classifier. To evaluate whether the classifier can accurately distinguish samples with accurate IR results, we use four metrics commonly used in classification problems to verify the performance of our Cross-Encoder based classifier and baselines, i.e., accuracy, precision, recall, and F1-score.

$$\begin{aligned}
 Accuracy &= \frac{TP + FN}{TP + FP + TN + FN} \\
 Precision &= \frac{TP}{TP + FP}, Recall = \frac{TP}{TP + FN} \\
 F1_score &= \frac{Precision \cdot Recall}{Precision + Recall}
 \end{aligned} \tag{13}$$

where TP/FP donates the number of *positive samples identified by the classifier that are/are not samples with accurate IR results*, and TN/FN donates the number of *negative samples identified by the classifier that are/are not samples with inaccurate IR results*.

4.5 Research Questions

We perform a large-scale comparative study to answer the following three research questions for evaluating our approach.

- **RQ 1 (Performance):** How does our approach compare to the commonly-used and state-of-the-art comment generation baselines?
- **RQ 2 (Accuracy of classification):** What is the accuracy of our Cross-Encoder based classifier?
- **RQ 3 (Generalizability):** Does our approach work with other NMT methods?

In RQ1, we set out to investigate the performance of generated comments of our proposed approach by comparing with seven state-of-the-art baselines (details are in Section 4.3.1). In RQ2, we evaluate whether the Cross-Encoder based classifier can effectively distinguish samples with accurate retrieved results by comparing with two baselines (details are in Section 4.3.2). In RQ3, we explore whether our approach is applicable for other neural comment generation approaches, i.e., still can obtain a significant improvement from dynamically combining with IR results.

5 RESULT ANALYSIS

5.1 RQ 1: Our Approach vs. Baselines

Experimental Method. To answer this research question, we compare our approach with comment generation baselines listed in Section 4.3. All baselines are trained on the FunCom training set. We compare generated comments of our approach and other baselines on the FunCom test set by four evaluation metrics described in Section 4.4.1.

Result. Table 1 shows the performance of our method compared to other comment generation baselines. Overall, our approach achieves the best performance on all evaluation metrics. Our approach achieves a 26% improvement on BLEU and a 7%-47% improvement on other metrics compared to DeepCom, the state-of-the-art neural-based approach, and achieves a 7% improvement on BLEU compared to Re²Com, the state-of-the-art IR+NMT approach.

From the table, we can see that the IR-based approach has a similar performance to neural-based approaches. The IR-based approach achieves 18.04 BLEU score, while neural-based approaches perform slightly better than it and achieve BLEU score range from 19.15 to 20.11. One of the possible reasons that the neural-based approaches and the IR-based approach perform similarly can be that the word distributions in the training and test datasets are different. Some custom identifiers in the test set samples may be rare or even absent from the training set, making it hard for the model to capture their information accurately [22].

For the two existing combinations of IR-based and neural-based approaches, i.e., Rencos and Re²Com, as we can see from the table, both could outperform IR-based and neural-based approaches. Specifically, Rencos achieves 19.86 BLEU score by fusing prediction results of the input code snippets with similar snippets. Re²Com achieves 23.69 BLEU scores by feeding the codes and comments of similar samples into the neural model. Our method achieves a higher 25.45 BLEU score by dynamically combining IR results and NMT results. In addition, both Rencos and Re²Com fail to improve the performance of the METEOR and ROUGE-L metrics significantly, but our approach achieves a significant improvement.

We have also conducted the Wilcoxon signed-rank test [45] ($p < 0.05$) to compare the performance of our approach and these baselines. The test result suggests that our approach achieves significantly better performance than baseline approaches in BLEU, METEOR, ROUGE-L, and CIDER.

Table 2: The performance of different classification algorithms

Approach	Classification Performance				Generated Comments
	Accuracy(%)	Precision(%)	Recall(%)	F1-score(%)	
lexical-level similarity	71.3	65.1	37.8	47.9	24.22
Siamese Network	68.9	59.2	34.4	43.5	23.5
Cross-Encoder	73.6	70.2	41.9	52.5	25.45

Our approach significantly outperforms the state-of-the-art comment generation baselines. The improvements on the IR-based approach, neural-based approach, and their combination are 41%, 26%, and 7% in terms of BLEU score, respectively.

5.2 RQ 2: Cross-Encoder vs. Other Classification Algorithms

Experimental Method. To answer this research question, we compare the Cross-Encoder based classifier with other classifier baselines listed in Section 4.3.2. Specifically, we apply these approaches on the test set labeled as described in Section 4.2.2 and use accuracy, precision, recall, and F1-score to measure the performance. In addition, we replace the Cross-Encoder based classifier of our approach with other classifier baselines, then use BLEU to measure the quality of the generated comments.

Result. The performance of each classification method is shown in Table 2. Overall, our approach (the Cross-Encoder based classifier) outperforms the two baselines on all the five metrics.

The first row of Table 2 shows the performance of the lexical-level similarity method (details are in Section 4.3.2), which achieves an accuracy of 71.3% in inferring whether the IR results are accurate. Its combined results achieve 24.22 BLEU score, which is better than Re²Com. Significant improvement can also be achieved even without training a classifier for comparison, which further validates that our idea of dynamically combining IR results with NMT results is indeed practical. However, the text-similarity-based approach also suffers the issues of false-positive as shown in Figure 2. To identify such false-positive samples, we use the Cross-Encoder, a semantic-based classifier, to more accurately predict whether the IR results are accurate.

The second row of Table 2 shows the performance of the Siamese network method (details are in Section 4.3.2). We train a Bi-LSTM network with strong expressive capability from scratch to determine semantics similarity. However, the Siamese network does not perform as well as expected; its performance is even worse than the lexical-level similarity method we showed above. One possible reason is that the model focuses on irrelevant features instead of the semantic gap between code snippet pair, leading to over-fitting and poor performance.

The third row of Table 2 shows the performance of our Cross-Encoder based classifier. Overall, our Cross-Encoder based classifier achieves the best performance on all metrics. The high accuracy (73.6%) and precision (70.2%) validate that it can help achieve our goal of filtering false-positive retrieval results, i.e., textually similar but semantically dissimilar. Besides, we can also see that the performance of the combined result increases with the increase of

Table 3: The performance (BLEU) of different NMT results combined with IR results. The percentages in parentheses indicate the relative improvement achieved by combining with IR results

Approach	NMT Only	Combined Result	Improvement
Rencos NMT Module	19.15	24.95	5.8 (30%)
attendgru	19.26	25.32	6.06 (31%)
ast-attendgru	19.73	25.34	5.61 (28%)
DeepCom	20.11	25.45	5.34 (26%)

accuracy of the classification, which suggests that the performance of our comment generation approach can be improved by better distinguishing samples with accurate IR result.

Our Cross-Encoder based classifier can accurately identify samples with accurate IR results. Besides, our idea of dynamically combining IR-based and neural-based approaches can outperform the state-of-the-art IR+NMT approaches even with the naive textual-similarity algorithm.

5.3 RQ 3: Generalizability

Experimental Method. Different neural models might generate different results, which can affect the generalizability of our approach. To evaluate the generalizability of our approach, we replace the DeepCom in our approach with three other neural-based baseline approaches (listed in Section 4.3). Then we measure the quality of generated comments with BLEU.

Result. Table 3 shows the performance of other neural-based approaches combined with IR results. Overall, after combining IR results, all three neural methods achieve better performance with 24.95-25.34 BLEU score. Specifically, Rencos NMT module, attendgru, and ast-attendgru can achieve relative improvements of 30%, 31%, and 28% from combining IR results, respectively, which are even higher than the relative improvement of DeepCom (26%). The above results fully demonstrate that the performance of our proposed approach remains stable across different neural approaches. Moreover, all the combined results outperform Re²Com, the current state-of-the-art IR+NMT method, which again validates the feasibility of our idea of dynamically combining IR results and NMT results.

The performance of our approach remains stable across different neural-based comment generation approaches.

6 DISCUSSION

6.1 Why Our Approach Performs Better?

To investigate why our proposed approach can achieve better performance, we partition the 90,908 samples in the test set into two sets, i.e., samples on which the IR-based approach performs better (IR-better samples) and samples on which the neural-based approach (DeepCom) performs better (NMT-better samples). Overall, there are 31,636 samples (34.8%) where the IR-based approach performs better, and 59,272 samples (65.2%) where the neural-based approach performs better. We then recalculate the performance (based on

Table 4: The performance (BLEU) of each approach on the IR-better samples and NMT-better samples

Approach	All	IR-better samples	NMT-better samples
	90908	31636 (34.8%)	59272 (65.2%)
Re ² Com Retrieve Module	18.04	39.55	5.25
DeepCom	20.11	20.86	19.58
Re ² Com	23.69	39.46	14.33
Our Method	25.45	37.5	18.0

BLEU) of the four methods in these two sets, i.e., Re²Com retrieve module (IR-based approach), DeepCom (neural-based approach), ReCom (IR+NMT approach), and our approach. The results are in Table 4.

From the table, we can see that for IR-better samples, the IR-based approach, i.e., Re²Com retrieve module, can directly leverage existing comments from similar samples in the corpus and achieves 39.55 BLEU score, which is almost twice as large as the score of the neural-based approach, i.e., DeepCom. For NMT-better samples, since no similar sample can be retrieved from the corpus, the IR-based approach performs poorly on these general samples and only achieves 5.25 BLEU score. In contrast, the neural-based approach can infer more accurate results by summarizing the code-to-comment pattern and achieves 19.58 BLEU score. The IR-based approach and the neural-based approach perform similarly on the whole test set, but their performance differs significantly on these two sets of samples. Thus combining the strengths of these two methods can achieve better performance.

By feeding information from the retrieved similar sample (code snippet and comment) to the neural model, the IR+NMT approach, i.e., Re²Com, performs better than the neural-based approach, i.e., DeepCom, on IR-better samples and achieves 39.46 BLEU score. However, on NMT-better samples, Re²Com only achieves 14.33 BLEU score, which is 27% lower than the score of DeepCom. The reason for such a performance degradation is that Re²Com can not accurately distinguish false-positive samples like Figure 2, thus incorrectly rely on the inaccurate retrieved information, i.e., the IR-based approach only achieves 5.25 BLEU score on NMT-better samples. Therefore, inaccurate retrieval information can lead to the degradation of the model’s generalization. In contrast, our approach directly distinguishes whether the retrieved result is accurate, which can help avoid the inaccurate retrieved information misleading the NMT to generate inaccurate comment. Thus our approach can outperform Re²Com on the NMT-better samples and the whole test set. Since the Cross-Encoder based classifier cannot perfectly predict whether the IR result is accurate, some samples incorrectly use inaccurate IR results as output or neglect accurate IR results. There is still a distance from the optimal performance of combing IR results and NMT results, i.e., achieving 39.55 BLEU score on IR-better samples and achieving 19.58 BLEU score on NMT-better samples.

6.2 Performance of Our Approach on An Alternative Dataset

To show the generalization of our approach, we further verify the performance of our method on another large-scale dataset, i.e., the

Table 5: The performance of each approach on the DeepCom dataset

Approach	BLEU	BLEU ₁	BLEU ₂	BLEU ₃	BLEU ₄
DeepCom	38.79	54.9	38.75	33.78	31.5
Re ² Com	50.21	61.83	50.6	46.29	43.89
Re ² Com Retrieval Module	55.28	65.93	55.27	51.69	49.59
Our Method	57.13	68.91	57.2	53.07	50.92

DeepCom dataset [17]. The DeepCom dataset was collected from GitHub’s Java repositories created from 2015 to 2016 and contained 445,812 code-comment pairs for training and 20,000 code-comment pairs for validation and testing.

We re-run our approach and the three baselines on the DeepCom Dataset, and the results are shown in Table 5. Overall, all four methods achieve outstanding performance on the DeepCom dataset, which quite different from their performance on the FunCom dataset. The main reason can be that the projects used in these two datasets are different, in which more code snippets and comments are reused among projects. The IR-based approach, Re²Com retrieval module, achieves 55.28 BLEU score on the test set, which implies that code reuse is more frequent on the projects collected by the DeepCom dataset. Thus the neural model can predict the samples in the test set more accurately due to the presence of similar samples in the training set. The neural-based approach, DeepCom, achieves 38.79 BLEU score, which seems to perform well, but it is even inferior to the naive IR-based method. By feeding codes and comments from retrieved similar samples, the IR+NMT method, Re²Com, achieves 50.21 BLEU score on the test set. However, the performance of Re²Com is still worse than the naive IR-based method, which implies that it fails to combine the strengths of the IR-based and NMT-based method on the DeepCom dataset. In contrast, our proposed approach, dynamically combining the generated results from DeepCom and IR-based approach, achieves 57.13 BLEU score on the test set, which successfully combines the strengths of the IR method and NMT method and achieves the best performance.

6.3 Effort Saved Comparing to The Existing Combination

Compared to the existing combination of IR- and NMT-based comment generation approaches, which use both the two models to generate a comment for each input sample, our approach dynamically selects the model to be used. To show the effort our method can save, we count the number of samples that do not need to run neural-based approaches to generate comments.

Specifically, our Cross-Encoder based classifier identifies 18,912 samples and 12,979 samples on the FunCom dataset and DeepCom dataset, respectively, that can be directly used for IR results. It implies that about 20% and 65% of the samples do not need to be fed into the NMT. Our approach can save the redundant effort of NMT predicting, making it faster than the current IR+NMT approach.

6.4 Threats to Validity

Internal Validity relates to the errors in the implementation of the baselines. To mitigate this issue, we directly use the public

available code of DeepCom [17], (ast-)attendgru [27], Re²Com [44], and Rencos [49] to implement baselines. Our experiments showed these baselines achieve comparable performance with the result reported in their papers.

External Validity is about the quality of our dataset. Different data sources can have significant different characteristics. Therefore, both our proposed approach and the baselines may perform differently on different datasets. In this paper, we only evaluate our proposed approach and baselines on two widely used datasets, i.e., DeepCom [17] and FunCom [27]. In our future work, we will experiment with other datasets.

Construct Validity relates to the suitability of our evaluation metrics. We use BLEU, ROUGE-L, METEOR, and CIDER to evaluate the generated comments of our approach and other baselines. These metrics mainly measure the gap between generated comments and ground truth in terms of textual similarity.

7 RELATED WORK

Comment generation. Code comment generation techniques can be divided into three types: manually-crafted templates [33, 40], IR-based [8, 9, 12, 13, 47], and neural models [16, 17, 19, 27, 44, 49].

Early studies leveraged manually-craft templates to generate comments automatically. Sridhara et al. [40] built the Software Word Usage Model (SWUM) to capture the meaning and relationship of terms in the source code, then organized them into readable comments using different predefined templates. Moreno et al. [33] used heuristic rules to capture critical information from the source code and further used them to generate comments.

Information retrieval (IR) techniques are also widely used in comment generation. One way is to provide extractive summaries of the source code, using IR techniques to extract keywords from the source code and compose them into term-based comments. Haiduc et al. [12, 13] treated each function of source code as a document and leveraged Vector Space Model and Latent Semantic Indexing (LSI) to extract relevant terms from source code, then organized selected terms into comments. Eddy et al. [8] took a similar idea and adopted a hierarchical topic model for improvement. Another way is directly use the existing comment of a similar sample. Since code reuse and cloning are common in software development, similar code snippets that use the same code fragments may be found in large project repositories (e.g., GitHub) or software Q&A sites (e.g., Stack Overflow). Edmund et al. [9, 47] retrieved the replicated samples from the corpus by code clone detection techniques.

More and more researchers have focused on neural-based methods, which train probabilistic models from large-scale source code in recent years. Iyer et al. [19] treated code to comment as an end-to-end translation problem and first introduced neural machine translation (NMT) into comment generation. They leveraged an attentional seq2seq model to translate code to comment, which used token embedding as the encoder and an LSTM layer as the decoder. Other researchers followed this way. Hu et al. [16] argued that treating code as natural language sequences may lose its syntactical information. They proposed a new structure-based traversal (SBT) method to flatten the AST into sequence and replaced code with it as the model input. Later they proposed another hybrid model [17] that simultaneously used codes and AST sequences for

prediction. LeClair et al. [27] also proposed a similar hybrid model but proved that the neural model also works with only the AST sequence known. The NMT-based method can automatically learn code to comment patterns from the corpus, which saves the manual effort to design features or templates and brings impressive generalization capability. The IR-based method may fail when there are no similar samples in the training set, but the NMT-based method can give more accurate answers.

IR-based Neural Comment Generation. The neural models are difficult to generate low-frequency tokens [25]. LeClair et al. [27] showed that about 21% of comments in their test set contained low-frequency words (frequency ≤ 100). However, only 7% generated results of their method contained low-frequency words. The IR-based methods leverage existing comments from similar samples, which may contain low-frequency words and project-specific information. Therefore, researchers have begun to combine IR-based methods with NMT-based methods by feeding information from similar samples (their codes only/ and comments) to assist neural models in better generating low-frequency words. Zhang et al. [49] proposed an approach that fused decoded results of the input code snippet and its similar code snippets, which were retrieved based on syntactic similarity and semantical similarity. Wei et al. [44] treated the existing comments of similar codes as exemplars, which can be reference examples for generating new comments. They introduced additional encoders to encode codes and comments from similar samples, then jointly trained model. To avoid the disturbance of inaccurate search results, both models decided the degree of using retrieved information based on the embedding similarity of the input and retrieved code snippets. The result shows that these methods can improve both the performance of generated comments and generating low-frequency words. However, both methods may be confused by false-positive samples like Figure 2. Without supervised learning, the input and retrieved code snippet of this example will yield similar embedding, making the model mistakenly believe that the retrieved results are accurate and wrongly rely on the inaccurate retrieved result, and leading to a decrease in generalization performance. In our work, we treat determining whether the retrieved result is accurate as a supervision task to distinguish false-positive retrieval results more accurately, and combine the IR-based and NMT-based methods in a dynamic manner to avoid the neural model over-rely on the retrieved information.

8 CONCLUSION

In this paper, we propose a dynamic approach to combine the strength of the IR-based and neural-based comment generation approaches. Specifically, given an input code snippet, we first use an IR-based technique to retrieve a similar code snippet from the corpus. Then we use a Cross-Encoder based classifier to decide the comment generation method to be used dynamically, i.e., if the retrieve similar code snippet is a true positive, we directly use the comment generated by IR-based approach. Otherwise, we input it to the neural-based approach to generate its comment. We have evaluated the effectiveness and generality of our approach on a large-scale Java dataset. The results show that our approach outperforms the state-of-the-art baselines by a significant margin.

REFERENCES

- [1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
- [2] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. 65–72.
- [3] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a "siamese" time delay neural network. *Advances in neural information processing systems* 6 (1993), 737–744.
- [4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).
- [5] Michael L Collard, Michael J Decker, and Jonathan I Maletic. 2011. Lightweight transformation and fact extraction with the srcML toolkit. In *2011 IEEE 11th international working conference on source code analysis and manipulation*. IEEE, 173–184.
- [6] Sergio Cozzetti B de Souza, Nicolas Anquetil, and Káthia M de Oliveira. 2005. A study of the documentation essential to software maintenance. In *Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*. 68–75.
- [7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
- [8] Brian P Eddy, Jeffrey A Robinson, Nicholas A Kraft, and Jeffrey C Carver. 2013. Evaluating source code summarization techniques: Replication and expansion. In *2013 21st International Conference on Program Comprehension (ICPC)*. IEEE, 13–22.
- [9] Wong Edmund. 2014. *Mining Question and Answer Sites for Automatic Comment Generation*. Master's thesis. University of Waterloo.
- [10] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocong Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).
- [11] David Gros, Hariharan Sezhiyan, Prem Devanbu, and Zhou Yu. 2020. Code to Comment "Translation": Data, Metrics, Baseline & Evaluation. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 746–757.
- [12] Sonia Haiduc, Jairo Aponte, and Andrian Marcus. 2010. Supporting program comprehension with source code summarization. In *2010 acm/ieee 32nd international conference on software engineering*, Vol. 2. IEEE, 223–226.
- [13] Sonia Haiduc, Jairo Aponte, Laura Moreno, and Andrian Marcus. 2010. On the use of automated text summarization techniques for summarizing source code. In *2010 17th Working Conference on Reverse Engineering*. IEEE, 35–44.
- [14] Sakib Haque, Aakash Bansal, Lingfei Wu, and Collin McMillan. 2021. Action Word Prediction for Neural Source Code Summarization. *arXiv preprint arXiv:2101.02742* (2021).
- [15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
- [16] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 200–20010.
- [17] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2020. Deep code comment generation with hybrid lexical and syntactical information. *Empirical Software Engineering* 25, 3 (2020), 2179–2217.
- [18] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436* (2019).
- [19] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2073–2083.
- [20] Mira Kajko-Mattsson. 2005. A survey of documentation practice within corrective maintenance. *Empirical Software Engineering* 10, 1 (2005), 31–55.
- [21] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. 2002. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering* 28, 7 (2002), 654–670.
- [22] Rafael-Michael Karampatsis, Hlib Babii, Romain Robbes, Charles Sutton, and Andrea Janes. 2020. Big code!= big vocabulary: Open-vocabulary models for source code. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1073–1085.
- [23] Miryung Kim, Vibha Sazawal, David Notkin, and Gail Murphy. 2005. An empirical study of code clone genealogies. In *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*. 187–196.
- [24] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
- [25] Philipp Koehn and Rebecca Knowles. 2017. Six challenges for neural machine translation. *arXiv preprint arXiv:1706.03872* (2017).
- [26] Douglas Kramer. 1999. API documentation from source code comments: a case study of Javadoc. In *Proceedings of the 17th annual international conference on Computer documentation*. 147–153.
- [27] Alexander LeClair, Siyuan Jiang, and Collin McMillan. 2019. A neural model for generating natural language summaries of program subroutines. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 795–806.
- [28] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. 2006. CP-Miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on Software Engineering* 32, 3 (2006), 176–192.
- [29] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.
- [30] Zhongxin Liu, Xin Xia, Ahmed E Hassan, David Lo, Zhenchang Xing, and Xinyu Wang. 2018. Neural-machine-translation-based commit message generation: how far are we?. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 373–384.
- [31] Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit. *arXiv preprint cs/0205028* (2002).
- [32] C. Lopes, S. Bajracharya, J. Oshser, and P. Baldi. 2010. UCI Source Code Data Sets. (2010). [http://www.ics.uci.edu/\\$simSlopes/datasets/](http://www.ics.uci.edu/$simSlopes/datasets/)
- [33] Laura Moreno, Jairo Aponte, Giriprasad Sridhara, Andrian Marcus, Lori Pollock, and K Vijay-Shanker. 2013. Automatic generation of natural language summaries for java classes. In *2013 21st International Conference on Program Comprehension (ICPC)*. IEEE, 23–32.
- [34] Sankar K Pal and Sushmita Mitra. 1992. Multilayer perceptron, fuzzy sets, classification. (1992).
- [35] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.
- [36] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. <http://arxiv.org/abs/1908.10084>
- [37] Alexander M Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685* (2015).
- [38] Mike Schuster and Kuldeep K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing* 45, 11 (1997), 2673–2681.
- [39] Kento Shimonaka, Soichi Sumi, Yoshiki Higo, and Shinji Kusumoto. 2016. Identifying auto-generated code by using machine learning techniques. In *2016 7th International Workshop on Empirical Software Engineering in Practice (IWSEPE)*. IEEE, 18–23.
- [40] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K Vijay-Shanker. 2010. Towards automatically generating summary comments for java methods. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*. 43–52.
- [41] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215* (2014).
- [42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762* (2017).
- [43] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. 2015. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4566–4575.
- [44] Bolin Wei, Yongmin Li, Ge Li, Xin Xia, and Zhi Jin. 2020. Retrieve and refine: exemplar-based neural comment generation. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 349–360.
- [45] Frank Wilcoxon, SK Katti, and Roberta A Wilcox. 1963. *Critical values and probability levels for the Wilcoxon rank sum test and the Wilcoxon signed rank test*. American Cyanamid Company Pearl River, NY.
- [46] Sam Wiseman and Alexander M Rush. 2016. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960* (2016).
- [47] Edmund Wong, Taiyue Liu, and Lin Tan. 2015. Clocom: Mining existing source code for automatic comment generation. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 380–389.
- [48] Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. 2016. Image captioning with semantic attention. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4651–4659.
- [49] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, and Xudong Liu. 2020. Retrieval-based neural source code summarization. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1385–1397.