RACE: Retrieval-Augmented Commit Message Generation

Ensheng Shi[a] Yanlin Wang[b,§,†] Wei Tao[c] Lun Du[d]
Hongyu Zhang[e] Shi Han[d] Dongmei Zhang[d] Hongbin Sun[a,§]
[a]Xi'an Jiaotong University [b]School of Software Engineering, Sun Yat-sen University
[c]Fudan University [d]Microsoft Research [e]The University of Newcastle
s1530129650@stu.xjtu.edu.cn, hsun@mail.xjtu.edu.cn
wangylin36@mail.sysu.edu.cn, wtao18@fudan.edu.cn
{lun.du, shihan, dongmeiz}@microsoft.com
hongyu.zhang@newcastle.edu.au

Abstract

Commit messages are important for software development and maintenance. Many neural network-based approaches have been proposed and shown promising results on automatic commit message generation. However, the generated commit messages could be repetitive or redundant. In this paper, we propose RACE, a new retrieval-augmented neural commit message generation method, which treats the retrieved similar commit as an exemplar and leverages it to generate an accurate commit message. As the retrieved commit message may not always accurately describe the content/intent of the current code diff, we also propose an *exemplar guider*, which learns the semantic similarity between the retrieved and current code diff and then guides the generation of commit message based on the similarity. We conduct extensive experiments on a large public dataset with five programming languages. Experimental results show that RACE can outperform all baselines. Furthermore, RACE can boost the performance of existing Seq2Seq models in commit message generation. Our data and source code are available at https://github.com/DeepSoftwareAnalytics/RACE.

1 Introduction

In software development and maintenance, source code is frequently changed. In practice, code changes are often documented as natural language commit messages, which summarize what (content) the code changes are or why (intent) the code is changed (Buse and Weimer, 2010; Cortes-Coy et al., 2014). High-quality commit messages are essential to help developers understand the evolution of software without diving into implementation details, which can save a large amount of time and effort in software development and maintenance (Dias et al., 2015; Barnett et al., 2015). However, it is difficult to write high-quality commit messages due to lack of time, clear motivation, or experienced skills. Even for seasoned developers, it still poses a considerable amount of extra workload to write a concise and informative commit message for massive code changes (Nie et al., 2021). It is also reported that around 14% of commit messages over 23,000 projects in SourceForge are left empty (Dyer et al., 2013). Thus, automatically generating commit messages becomes an important task.

Over the years, many approaches have been proposed to automatically generate commit messages. Early studies (Shen et al., 2016; Cortes-Coy et al., 2014) are mainly based on predefined rules or templates, which may not cover all situations or comprehensively infer the intentions behind code changes. Later, some studies (Liu et al., 2018; Huang et al., 2017, 2020) adopt information retrieval (IR) techniques to reuse commit messages of similar code changes. They can take advantage of similar examples, but the reused commit messages might not correctly describe the content/intent of the current code change. Recently, some Seq2Seq-based neural network models (Loyola et al., 2017; Jiang et al., 2017; Xu et al., 2019; Liu et al., 2019; Jung, 2021) have been proposed to understand code diffs and generate the high-quality commit messages. These approaches show promising performance, but they tend to generate high-frequency and repetitive tokens and the generated commit messages have the problem of insufficient information and poor readability (Wang et al., 2021a; Liu et al., 2018). Some studies (Liu et al., 2020; Wang et al., 2021a) also explore the combination of neural-based and IR-based techniques. Liu et al. (2020) propose an approach to rank the retrieved commit message (obtained by a simple IR-based model) and the generated commit message (ob-

[§]Yanlin Wang and Hongbin Sun are the corresponding authors.

[†]Work done during the author's employment at Microsoft Research Asia

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 5520–5530
December 7-11, 2022

tained by a neural network model). Wang et al. (2021a) propose to use the similar code diff as auxiliary information in the inference stage, while the model is not trained to learn how to effectively utilize the information of retrieval results. Therefore, both of them fail to take advantage of the information of retrieved similar results well.

In this paper, we propose a novel model **RACE** (Retrieval-Augmented Commit mEssage generation), which retrieves a similar commit message as an exemplar, guides the neural model to learn the content of the code diff and the intent behind the code diff, and generates the readable and informative commit message. The key idea of our approach is retrieval and augmentation. Specifically, we first train a code diff encoder to learn the semantics of code diffs and encode the code diff into high-dimensional semantic space. Then, we retrieve the semantically similar code diff paired with the commit message on a large parallel corpus based on the similarity measured by vectors' distance. Next, we treat the similar commit message as an exemplar and leverage it to guide the neural-based models to generate an accurate commit message. However, the retrieved commit messages may not accurately describe the content/intent of current code diffs and may even contain wrong or irrelevant information. To avoid the retrieved samples dominating the processing of commit message generation, we propose an *exemplar guider*, which first learns the semantic similarity between the retrieved and current code diff and then leverages the information of the exemplar based on the learned similarity to guide the commit message generation.

To evaluate the effectiveness of RACE, we conduct experiments on a large-scale dataset MCMD (Tao et al., 2021) with five programming language (Java, C#, C++, Python and JavaScript) and compare RACE with 11 state-of-the-art approaches. Experimental results show that: (1) RACE significantly outperforms existing state-of-the-art approaches in terms of four metrics (BLUE, Meteor, Rouge-L and Cider) on the commit message generation. (2) RACE can boost the performance of existing Seq2Seq models in commit message generation. For example, it can improve the performance of NMTGen (Loyola et al., 2017), CommitBERT (Jung, 2021), CodeT5-small (Wang et al., 2021b) and CodeT5-base (Wang et al., 2021b) by 43%, 11%, 15%, and 16% on average in terms of BLEU, respectively. In addition, we also conduct human evaluation to confirm the effectiveness of RACE.

We summarize the main contributions of this paper as follows:

- We propose a retrieval-augmented neural commit message generation model, which treats the retrieved similar commit as an exemplar and leverages it to guide neural network model to generate informative and readable commit messages.
- We apply our retrieval-augmented framework to four existing neural network-based approaches (NMTGen, CommitBERT, CodeT5-small, and CodeT5-base) and greatly boost their performance.
- We perform extensive experiments including human evaluation on a large multi-programming-language dataset and the results confirm the effectiveness of our approach over state-of-the-art approaches.

2 Related Work

Code intelligence, which leverages machine learning especially deep learning-based method to understand source code, is an emerging topic and has obtained the promising results in many software engineering tasks, such as code summarization (Zhang et al., 2020; Shi et al., 2021a, 2022b; Wang et al., 2020) and code search (Gu et al., 2018; Du et al., 2021; Shi et al., 2022a). Among them, commit message generation plays an important role in the software evolution.

In early work, information retrieval techniques are introduced to commit message generation (Liu et al., 2018; Huang et al., 2017, 2020). For instance, ChangeDoc (Huang et al., 2020) retrieves the most similar commits according to the syntax or semantics in the code diff and reuses commit messages of similar code diffs. **NNGen** (Liu et al., 2018) is a simple yet effective retrieval-based method using the nearest neighbor algorithm. It firstly recalls the top-k similar code diffs in the parallel corpus based on cosine similarity between bag-of-words vectors of code diffs. Then select the most similar result based on BLEU scores between each of them (top-k results) and the input code diff. These approaches can reuse similar examples and the reused commit messages are usually readable and understandable.

Recently, many neural-based approaches (Loyola et al., 2017; Jiang et al., 2017; Xu et al., 2019;

Liu et al., 2019, 2020; Jung, 2021; Dong et al., 2022; Nie et al., 2021; Wang et al., 2021a) have been used to learn the semantic of code diffs and translate them into commit messages. For example, **NMTGen** (Loyola et al., 2017) and **CommitGen** (Jiang et al., 2017) treat the code diffs as plain texts and adopt the Seq2Seq neural network with different attention mechanisms to translate them into commit messages. **CoDiSum** (Xu et al., 2019) extracts both code structure and code semantics from code diffs and jointly models them with a multi-layer bidirectional GRU to better learn the representations of code diffs. **PtrGNCMsg** (Liu et al., 2019) incorporates the pointer-generator network into the Seq2Seq model to handle out-of-vocabulary (OOV) words. **CommitBERT** leverage CodeBERT (Feng et al., 2020), a pre-trained language model for source code, to learn the semantic representations of code diffs and adopt a Transformer-based (Vaswani et al., 2017) decoder to generate the commit message. These approaches show promising results on the generation of commit messages.

Recently, introducing retrieved relevant results into the training process has been found useful in most generation tasks (Lewis et al., 2020; Yu et al., 2021; Wei et al., 2020). Some studies (Liu et al., 2020; Wang et al., 2021a) also explore the combination of neural-based models and IR-based techniques to generate commit messages. **ATOM** (Liu et al., 2020) ensembles the neural-based model and the IR-based technique through the hybrid ranking. Specifically, it uses BiLSTM to encode ASTs paths extracted from ASTs of code diffs and adopt a decoder to generate commit messages. It also uses TF-IDF technique to represent code diffs as vectors and retrieves the most similar commit message based on cosine similarity. The generated and retrieved commit messages are finally prioritized by a hybrid ranking module. **CoRec** (Wang et al., 2021a) is also a hybrid model and only considers the retrieved result during the inference. Specifically, at the training stage, they use an encoder-decoder neural model to encode the input code diffs by an encoder and generate commit messages by a decoder. At the inference stage, they first use the trained encoder to retrieve the most similar code diff from the training set. Then they reuse a trained encoder-decoder to encode the input and retrieved code diff, combine the probability distributions (obtained by two decoders) of each word, and generate the final commit message step by step. In summary, ATOM does not learn to refine the retrieved results or the generated results, and CoRec is not trained to utilize the information of retrieval results. Therefore, both of them fail to take full advantage of the retrieved similar results. In this paper, we treat the retrieved similar commit as an exemplar and train the model to leverage the exemplar to enhance commit message generation.

3 Proposed Approach

The overview of RACE is shown in Figure 1. It includes two modules: retrieval module and generation module. Specifically, RACE firstly retrieves the most semantically similar code diff paired with the commit message from the large parallel training corpus. The semantic similarity between two code diffs is measured by the cosine similarity of vectors obtained by a code diff encoder. Next, RACE treats the retrieved commit message as an example and uses it to guide the neural network to generate an understandable and concise commit message.

3.1 Retrieval module

In this module, we aim to retrieve the most semantically similar result. Specifically, we first train an encoder-decoder neural network on the large commit message generation dataset. The encoder is used to learn the semantics of code diffs and encode code diffs into a high-dimension semantic space. Then we retrieve the most semantically similar code diff paired with the commit message from the large parallel training corpus. The semantic similarity between two code diffs is measured by the cosine similarity of vectors obtained by a well-trained code diff encoder.

Recently, encoder-decoder neural network models (Loyola et al., 2017; Jiang et al., 2017; Jung, 2021), which leverage an encoder to learn the semantic of code diff and employ a decoder to generate the commit message, have shown their superiority in the understanding of code diffs and commit messages generation. To enable the code diff encoder to understand the semantics of code diffs, we train it with a commit message generator on a large commit message generation dataset, which consists of about 0.9 million <code diff , commit message> pairs.

To capture long-range dependencies (e.g. a variable is initialized before the changed line) and more contextual information of code diffs, we em-

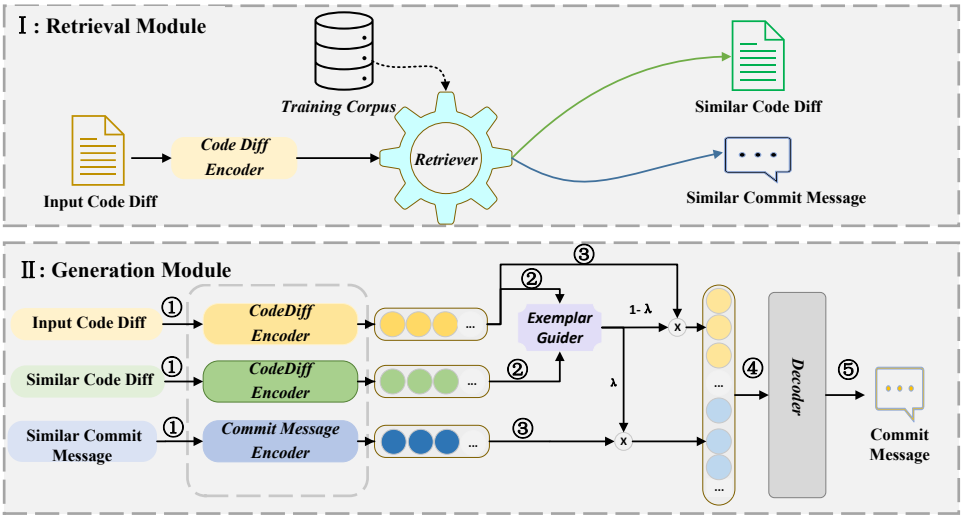


Figure 1: The architecture of RACE. It includes two modules: retrieval module and generation module. The retrieval module is used to retrieve the most similar code diff and commit message. The generation module leverages the retrieved result to enhance the performance of neural network models.

ploy a Transformer-based encoder to learn the semantic representations of input code diffs. As shown in Figure 1, a Transformer-based encoder is stacked with multiple encoder layers. Each layer consists of four parts, namely, a multi-head self-attention module, a relative position embedding module, a feed forward network (FFN) and an add & norm module. In b-th attention head, the input $\mathbf{X^b} = \left(\mathbf{x_1^b}, \mathbf{x_2^b}, ..., \mathbf{x_l^b}\right)$ (where $\mathbf{X^b} = \mathbf{X}[(b-1) * head_{dim} : b * head_{dim}]$, $\mathbf{X}$ is the sequence of code diff embedding, $head_{dim}$ is the dimension of each head and l is the input sequence length.) is transformed to $(\mathbf{Head}^b = \mathbf{head_1^b}, \mathbf{head_2^b}, ..., \mathbf{head_l^b})$ by:

$$\begin{aligned} \mathbf{head_i^b} &= \sum_{j=1}^{l} \alpha_{ij} \left(\mathbf{W_V x_j^b} + \mathbf{p_{ij}^V}\right) \\ e_{ij} &= \frac{\left(\mathbf{W_Q x_i^b}\right)^T \left(\mathbf{W_K x_j^b} + \mathbf{p_{ij}^K}\right)}{\sqrt{d_k}} \end{aligned} \tag{1}$$

where $\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{ik}}$, $\mathbf{W_Q}$, $\mathbf{W_K}$ and $\mathbf{W_V}$ are learnable matrix for queries, keys and values. d_k is the dimension of queries and keys; $\mathbf{p_{ij}^K}$ and $\mathbf{p_{ij}^V}$ are relative positional representations for positions i and j.

The outputs of all heads are concatenated and then fed to the FFN modules which is a multi-layer perception. The add & norm operation are employed after the multi-head attention and FFN modules. The calculations are as follows:

$$\begin{aligned} \mathbf{Head} &= Concat\left(\mathbf{Head^1}, \mathbf{Head^d}, \mathbf{Head^B}\right) \\ \mathbf{Hid} &= add\ \&\ norm\left(\mathbf{Head}, \mathbf{X}\right) \\ \mathbf{Enc} &= add\ \&\ norm\left(\mathbf{FFN}\left(\mathbf{Hid}\right), \mathbf{Hid}\right) \end{aligned} \tag{2}$$

where $add\ \&\ norm\left(\mathbf{A_1}, \mathbf{A_2}\right) = LN\left(\mathbf{A_1} + \mathbf{A_2}\right)$, B is the number of heads and LN is layer normalization. The final output of encoder is sent to Transformer-based decoder to generate the commit message step by step. We use cross-entropy as loss function and adopt AdamW (Loshchilov and Hutter, 2019) to optimize the parameters of the code diff encoder and the decoder at the top of Figure 1.

Next, the retrieval module is used to retrieve the most similar result from a large parallel training corpus. We firstly use the above code diff encoder to map code diffs into a high-dimensional latent space and retrieve the most similar example based on cosine similarity.

Specifically, after being trained in the commit message generation dataset, the code diff encoder can capture the semantic of code diff well. We use well-trained code diff encoder following a mean-pooling operation to map the code diff into a high dimensional space. Mathematically, given the input code diff embedding $\mathbf{X} = (\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_l})$, the code diff encoder can transformed them to $\mathbf{Enc} = (\mathbf{enc_1}, \mathbf{enc_2}, ..., \mathbf{enc_l})$. Then we obtain the semantic vector of the code diff by pooling operation:

$$\mathbf{vec} = pooling(\mathbf{Enc}) = mean\left(\mathbf{enc_1}, \mathbf{enc_2}, ..., \mathbf{enc_l}\right) \tag{3}$$

where mean is a dimension-wise average operation. We measure the similarity of two code diffs by cosine similarity of their semantic vectors and retrieve the most similar code diff paired with the commit message from the parallel training corpus. For each

code diff, we return the first-ranked similar result. But, for the code diff in the training dataset, we return the second-ranked similar result because the first-ranked result is itself.

3.2 Generation module

As shown at the bottom of Figure 1, in the generation module, we treat the retrieved commit message as an exemplar and leverage it to guide the neural network model to generate an accurate commit message. Our generation module consists of three components: three encoders, an exemplar guider, and a decoder.

First, following Equation 1, 2, three Transformer-based encoders are adopted to obtain the representations of the input code diff ($\mathbf{Enc^d} = \mathbf{enc}_1^d, \mathbf{enc}_2^d, ..., \mathbf{enc}_l^d$), the similar code diff ($\mathbf{Enc^s} = \mathbf{enc}_1^s, \mathbf{enc}_2^s, ..., \mathbf{enc}_m^s$), and similar commit message ($\mathbf{Enc^m} = \mathbf{enc}_1^m, \mathbf{enc}_2^m, ..., \mathbf{enc}_n^m$) (step ① in Figure 1), where subscripts l, m, n are the length of the input code diff, the similar code diff, and the similar commit message, respectively.

Second, since the retrieved similar commit messages may not always accurately describe the content/intent of the input code diffs even express totally wrong or irrelevant semantics. Therefore, we propose an *exemplar guider* which first learns the semantic similarity between the retrieved and input code diff and then leverages the information of the similar commit messages based on the learned similarity to guide the commit message generation (step ②). Mathematically, *exemplar guider* calculate the semantic similarity (λ) between the input code diff and the similar code diff based on their representation $\mathbf{Enc}_l^d$ and $\mathbf{Enc}_m^s$ (step ② and ③):

$$\lambda = \sigma(\mathbf{W_s}[mean(\mathbf{Enc}^d), mean(\mathbf{Enc}^s)]) \quad (4)$$

where σ is the sigmoid activation function, $\mathbf{W_s}$ is a learnable matrix, and $mean$ is a dimension-wise average operation.

Third, we weight representations of code diff and similar commit message by $1-\lambda$ and λ, respectively and then concatenate them to obtain the final input encoding.

$$\mathbf{Enc^{dm}} = [(1-\lambda) * \mathbf{Enc^d} : \lambda * \mathbf{Enc^s}] \quad (5)$$

Finally, we use a Transformer-based decoder to generate the commit message. The decoder consists of multiply decoder layer and each layers includes a masked multi-head self-attention, a multi-head cross-attention module, a FFN module and an add & norm module. Different from multi-head self-attention module in the encoder, in terms of one token, masked multi-head self-attention in the decoder can only attend to the previous tokens rather than the before and after context. In b-th cross-attention layer, the input encoding ($\mathbf{Enc^{dm}} = \left(\mathbf{enc_1^{dm}}, \mathbf{enc_2^{dm}}, ..., \mathbf{enc_{l+m}^{dm}}\right)$) is queried by the output of the preceding commit message representations $\mathbf{Msg} = (\mathbf{msg_1}, ..., \mathbf{msg_t})$ obtained by masked multi-head self-attention module.

$$\begin{aligned} Dec_{head_i^b} &= \sum_{j=1}^{l+m} \alpha_{ij} \left(\mathbf{W_V^{Dec}} \mathbf{enc_j^b}\right) \\ Dec_{e_{ij}} &= \frac{(\mathbf{W_Q^{Dec}} \mathbf{msg_j^b})^T (\mathbf{W_K^{Dec}} \mathbf{enc_i^b})}{\sqrt{d_k}} \end{aligned} \quad (6)$$

where $\alpha_{ij} = \frac{\exp Dec_{e_{ij}}}{\sum_{k=1}^{n} \exp Dec_{e_{ik}}}$, $\mathbf{W_Q^{Dec}}$, $\mathbf{W_K^{Dec}}$ and $\mathbf{W_V^{Dec}}$ are trainable projection matrices for queries, keys and values of the decoder layer. $\mathbf{t}$ is the length of preceding commit message.

Next, we use Equation 2 to obtain the hidden states of each decoder layer. In the last decoder layers, we employ a MLP and softmax operator to obtain the generation probability of each commit message token on the vocabulary. Then we use the cross-entropy as the loss function and apply AdamW for optimization.

Language	Training	Validation	Test
Java	160,018	19,825	20,159
C#	149,907	18,688	18,702
C++	160,948	20,000	20,141
Python	206,777	25,912	25,837
JavaScript	197,529	24,899	24,773

Table 1: Statistics of the evaluation dataset.

4 Experimental Setup

4.1 Dataset

In our experiment, we use a large-scale dataset MCMD (Tao et al., 2021) with five programming languages (PLs): Java, C#, C++, Python and JavaScript. For each PL, MCMD collects commits from the top-100 starred repositories on GitHub and then filters the redundant messages (such as rollback commits) and noisy messages defined in Liu et al. (2018). Finally, to balance the size of data, they randomly sample and retain 450,000 commits for each PL. Each commit contains the code diff, the commit message, the name of the repository,

and the timestamp of commit, etc. To reduce the noise data in the dataset, we further filter out commits that contain multiple files or files that cannot be parsed (such as `.jar`, `.ddl`, `.mp3`, and `.apk`).

4.2 Data pre-processing

The code diff in MCMD are based on line-level code change. To obtain more fine-grained code change, following previous study (Panthaplackel et al., 2020), we use a sequence of span of token-level change actions to represent the code diff. Each action is structured as <action> span of tokens <action end>. There are four <action> types, namely, <keep>, <insert>, <delete>, and <replace>. <keep> means that the span of tokens are unchanged. <insert> means that adding span of tokens. <delete> means that deleting span of tokens. <replace> means that the span of tokens in the old version that will be replaced with different span of tokens in the new version. Thus, we extend <replace> to <replace old> and <replace new> to indicate the span of old and new tokens, respectively. We use difflib [1] to extract the sequence of code change actions.

4.3 Hyperparameters

We follow (Tao et al., 2021) to set the maximum lengths of code diff and commit message to 200 and 50, respectively. We use the weight of the encoder of CodeT5-base (Wang et al., 2021b) to initialize the code diff encoders and use the decoder of CodeT5-base to initialize the decoder in Figure 1. The original vocabulary sizes of CodeT5 is 32,100. We add nine special tokens (<keep>, <keep_end>, <insert>, <insert_end>, <delete>, <delete_end>, <replace_old>, <replace_new>, and <replace_end>) and the vocabulary sizes of code and queries become 32109. For the optimizer, we use AdamW with the learning rate 2e-5. The batch size is 32. The max epoch is 20. In addition, we run the experiments 3 times with random seeds 0,1,2 and display the mean value in the paper. The experiments are conducted on a server with 4 GPUs of NVIDIA Tesla V100 and it takes about 1.2 hours each epoch.

4.4 Evaluation metrics

We evaluate the quality of the generated messages using four metrics: BLEU (Papineni et al., 2002), Meteor (Banerjee and Lavie, 2005), Rouge-L (Lin, 2004), and Cider (Vedantam et al., 2015). These metrics are prevalent metrics in machine translation, text summarization, and image captioning. There are many variants of BLEU being used to measure the generated message, We choose B-Norm (the BLEU result in this paper is B-Norm), which correlates with human perception the most (Tao et al., 2021). The detailed metrics calculation can be found in Appendix.

4.5 Baselines

We compare RACE with four end-to-end neural-based models, two IR-based methods, two hybrid approaches which combine IR-based techniques and end-to-end neural-based methods, and three pre-trained-based models. Four end-to-end neural-based models include CommitGen (Jiang et al., 2017), CoDiSum (Xu et al., 2019), NMTGen (Loyola et al., 2017), PtrGNCMsg (Liu et al., 2019) and ATOM (Liu et al., 2020). They all train models from scratch. Two IR-based methods are NNGen (Liu et al., 2018) and Lucene (Apache, 2011), they retrieve the similar code diff based on different similarity measurements and reuse the commit message of the similar code diff as the final result. CoRec and ATOM are all hybrid models which combine the neural-based models and IR-based techniques. Three pre-trained models are CommitBERT, CodeT5-small, and CodeT5-base. They are pre-trained on the large parallel code and natural language corpus and fine- tuned on the commit message generation dataset. All baselines except Lucene, CodeT5-small and CodeT5-base are introduced in Section 2. Lucene is a traditional IR baseline, which uses TF-IDF to represent a code diff as a vector and searches the similar code diff based on the cosine similarity between two vectors. CodeT5-small and CodeT5-base are source code pre-trained models and have achieved promising results in many code-related tasks (Wang et al., 2021b). We fine-tune them on MCMD as strong baselines. In addition, we only evaluate ATOM on Java dataset as the current implementation of ATOM only supports Java.

5 Experimental Results

5.1 How does RACE perform compared with baseline approaches?

To evaluate the effectiveness of RACE, we conduct the experiment by comparing it with the 11

[1] https://docs.python.org/3/library/difflib.html

	Model	Java				C#				C++				Python				JavaScript			
		BLEU	Met.	Rou.	Cid.	BLEU	Met.	Rou.	Cid.	BLEU	Met.	Rou.	Cid.	BLEU	Met.	Rou.	Cid.	BLEU	Met.	Rou.	Cid.
IR-based	NNGen	19.41	12.40	25.15	1.23	22.15	14.77	26.46	1.55	13.61	9.39	18.21	0.73	16.06	10.91	21.69	0.92	18.65	12.50	24.45	1.21
	Lucene	15.61	10.56	19.43	0.94	20.68	13.34	23.02	1.36	13.43	8.81	16.78	0.67	15.16	9.63	18.85	0.85	17.66	11.25	21.75	1.02
End-to-end	CommitGen	14.07	7.52	18.78	0.66	13.38	8.31	17.44	0.63	11.52	6.98	16.75	0.45	11.02	6.43	16.64	0.42	18.67	11.88	24.10	1.08
	CoDiSum	13.97	6.02	16.12	0.39	12.71	5.56	14.40	0.36	12.44	6.00	14.39	0.42	14.61	8.59	17.02	0.42	11.22	5.32	13.26	0.28
	NMTGen	15.52	8.91	21.13	0.86	12.71	8.11	17.16	0.62	11.57	7.06	17.46	0.51	11.41	7.18	18.43	0.48	18.22	12.07	24.43	1.12
	PtrGNCMsg	17.71	11.33	24.32	0.99	15.98	10.18	21.16	0.83	14.06	9.63	20.17	0.63	15.89	11.36	23.49	0.76	20.78	14.52	27.87	1.29
Hybrid	ATOM	16.42	11.66	22.67	0.91	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/	/
	CoRec	18.51	11.26	24.78	1.13	18.41	11.70	23.73	1.12	14.02	8.63	20.10	0.72	15.09	9.60	22.35	0.80	21.30	13.84	27.53	1.40
Pre-trained	CommitBERT	22.32	12.63	28.03	1.42	20.67	12.31	25.76	1.25	16.16	10.05	19.90	0.94	17.29	11.31	22.36	1.01	23.40	15.64	30.51	1.54
	CodeT5-small	22.28	14.16	29.71	1.37	18.92	11.71	24.95	1.05	16.08	11.19	21.60	0.79	17.49	12.46	24.65	0.90	21.97	14.48	28.65	1.42
	CodeT5-base	22.76	14.57	30.23	1.43	22.21	14.51	29.08	1.33	16.73	11.69	22.86	0.85	17.99	12.74	25.27	0.96	22.87	15.12	29.81	1.50
Ours	**RACE**	**25.66** ↑13%	**15.46** ↑6%	**32.02** ↑6%	**1.76** ↑23%	**26.33** ↑19%	**16.37** ↑13%	**31.31** ↑8%	**1.84** ↑38%	**19.13** ↑14%	**12.55** ↑7%	**24.52** ↑7%	**1.14** ↑34%	**21.79** ↑21%	**14.68** ↑15%	**28.35** ↑12%	**1.40** ↑46%	**25.55** ↑12%	**16.31** ↑8%	**31.79** ↑7%	**1.84** ↑23%
Ablation	RACE -Guider	23.37	13.98	30.01	1.53	21.33	13.56	27.33	1.31	17.43	12.10	22.03	0.95	19.44	13.89	26.4	1.01	23.39	15.64	30.51	1.54

Table 2: Comparison of RACE with baselines under four metrics on five programming languages. Met., Rou., and Cide. are short for Meteor, Rouge-L, and Cider, respectively. All results are statistically significant (with $p < 0.01$).

baselines including two IR-based approaches, four end-to-end neural-based approaches, two hybrid approaches, and three pre-train-based approaches in terms of four evaluation metrics. The experimental results are shown in Table 2.

We can see that IR-based models NNGen and Lucene generally outperform end-to-end neural models on average in terms of four metrics. It indicates that retrieved similar results can provide important information for commit message generation. CoRec, which combines the IR-based method and neural method, performs better than NNGen on C++ and JavaScript dataset but lower than NNGen on Java, C# and Python. This is because CoRec only leverages the information similar code diff at the inference stage. ATOM, which priorities the generated result of the neural-based model and retrieved result of the IR-based method, also outperforms the IR-based approach Lucene and three neural-based models CommitGen, CoDiSum, and NMTGen. Three pre-trained-based approaches outperform other baselines in terms of four metrics on average. CodeT5-base performs best among them on average. Our approach performs the best among all approaches on 5 programming languages in terms of four metrics. This is because RACE treats the retrieved similar commit message as an exemplar and leverages it to guide the neural network model to generate an accurate commit message.

We also give an example of commit messages generated by our approach and the baselines in Figure 2. IR-based methods NNGen and Lucene can retrieve semantically similar but not completely correct commit message. Specifically, retrieved commit messages contain not only the important semantic ("Filter out unavailable databases") of the current code diff but also the extra information ("Revert"). Neural network models generally capture the action of "add" but fail to further understand the intend of the code diff. The hybrid model CoRec cannot generate the correct commit message either. Our model treats the retrieved result (Revert "Filter out unavailable databases") as an exemplar, and guides the neural network model to generate the correct commit message.

5.2 What is the effectiveness of exemplar guider?

We conduct the ablation study to verify the effectiveness of *exemplar guider* module. Specifically, as shown at the bottom of Figure 1, we directly concatenated the representations of retrieved results and fed them to the decoder to generate commit messages without using the *exemplar guider*. As shown at the bottom of the Table 2, we can see that the performance of the ablated model (RACE -Guide) degrades in all programming languages in terms of four metrics. It demonstrates the effectiveness of our *exemplar guider*.

5.3 What is the performance when we reteieve k relevant commits?

We also conduct experiments to recall k (k=1, 3, 5, 7, 9) most relevant commits to augment the generation model. Specifically, as shown in Figure 1 the relevance of the code diffs is measured by the cosine similarity their semantic vectors obtained by Equation 3. Then retrieved k relevant commits are encoded and fed to the exemplar guidar to obtain

Code Diff

```
apache: superset/views/core.py
 ...
+ class DatabaseFilter(SupersetFilter):
+     def apply(self, query, func):
+         if self.has_all_datasource_access():
+             return query
+         perms = self.get_view_menus('database_access')
+         return query.filter(self.model.perm.in_(perms))
 ...
+     base_filters = [['perm', DatabaseFilter, lambda:[]]]
```

Reference *Filter out unavailable databases*

Baselines	
NNGen	Revert " Filter out unavailable databases"
Lucene	Revert " filter out unavailable databases "
CommitGen	Merge pull request from mistercrunch / UNK
NMTGen	Add <unk> to <unk>
PtrGNCMsg	Add support for dashboards in database
CoRec	Remove <unk>
CommitBERT	Add DatabaseFilter ( )
CodeT5-small	[database] Add databasefilter to filter all users
CodeT5-base	[hotfix] Adding databasefilter to core.py
RACE	Stage I : Revert " Filter out unavailable databases " Stage II : **Filter out unavailable databases**

Figure 2: An example of generated commit messages. Reference is the developer-written commit message. The results of our approach in stage I and II are returned by the retrieved module and generation module, respectively.

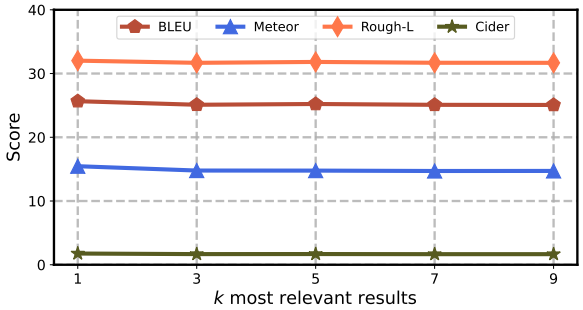
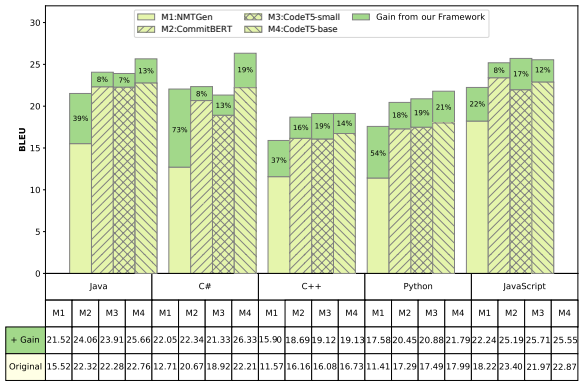


Figure 3: Performance of models augemented with k retrieved relevant commits.



	Java				C#				C++				Python				JavaScript			
	M1	M2	M3	M4	M1	M2	M3	M4	M1	M2	M3	M4	M1	M2	M3	M4	M1	M2	M3	M4
+ Gain	21.52	24.06	23.91	25.66	22.05	22.34	21.33	26.33	15.90	18.69	19.12	19.13	17.58	20.45	20.88	21.79	22.24	25.19	25.71	25.55
Original	15.52	22.32	22.28	22.76	12.71	20.67	18.92	22.21	11.57	16.16	16.08	16.73	11.41	17.29	17.49	17.99	18.22	23.40	21.97	22.87

Figure 4: Performance gains on four models. The original performance of the models are in yellow and gains from our framework are in green. The percentage value in each bar is the rate of improvement.

semantic similarities by Equation 4, respectively. Finally, we weight representations of code diff and similar commit messages according to the semantic similarities and feed them to the decoder to generate commit messages step by step. The experimental results are shown in Figure 3. We can see that the performance is generally stable on different k. In our future work, we will continue to study alternatives on leveraging the information of the retrieved results, e.g., how many commits to retrieve and how to model the corresponding information.

5.4 Can our framework boost the performance of existing models?

We further study whether our framework can enhance the performance of the existing Seq2Seq neural network model in commit message generation. Therefore, we adapt our framework to four Seq2Seq-based models, namely NMTGen (M1), CommitBERT (M2), CodeT5-small (M3) and CodeT5-base (M4). Specifically, we use the encoder of these models as our code diff encoder and obtain the high-dimensional semantic vectors in the retrieval module (Figure 1). In the generation module, we use the encoder of their models to encode input code diffs, similar code diffs, and similar commit messages. We also use the decoder of their models to generate commit messages.

The experimental results are shown in Figure 4, we present the performance of four original models (yellow) and gains (green) from our framework on five programming languages in terms of BLEU [2] score. Overall, we can see that our framework can improve the performance of all four neural models in all programming languages. Our framework can improve the performance of the original model from 7% to 73%. Especially, after applying our framework, the performance of NMTGen has more than 20% improvement on all programming languages. In addition, Our framework can *boost* the performance of NMTGen on BLUE, Meteor, Rouge-L, and Cider by 43%, 49%, 33%, and 61% on average, *boost* CommitBERT by 11%, 9%, 11%, and 12%, *boost* CodeT5-small by 15%, 14%, 11%, and 26%, and *boost* CodeT5-base by 16%, 10%, 8%, and 32% [3].

[2]We show results of other three metrics in Appendix due to space limitation. Our conclusions also hold.

[3]The result can be found in 1-4 of Appendix

Model	Informativeness	Conciseness	Expressiveness
CommitBERT	1.22 (±1.02)	2.03 (±1.04)	2.46 (±0.99)
NNGen	1.03 (±1.00)	1.74 (±1.01)	2.36 (±0.95)
NMTGen	0.74 (±0.92)	1.56 (±0.93)	2.11 (±0.94)
CoRec	1.05 (±1.09)	1.80 (±1.05)	2.43 (±0.88)
RACE	**2.49** (±1.10)	**3.08** (±0.96)	**2.85** (±0.84)

Table 3: Results of human evaluation (standard deviation in parentheses).

5.5 Human evaluation

We also conduct a human evaluation by following the previous works (Moreno et al., 2013; Panichella et al., 2016; Shi et al., 2021b) to evaluate the semantic similarity of the commit message generated by RACE and four baselines NNGen, NMTGen, CommitBERT, and CoRec. The four baselines are IR-based, end-to-end neural network-based, hybrid, and pre-trained-based approaches, respectively. We randomly choose 50 code diff from the testing sets and their commit message generated by four approaches. Finally, we sample 250 <code diff , commit message> pairs to score. Specifically, we invite 4 volunteers with excellent English ability and more than three years of software development experience. Each volunteer is asked to assign scores from 0 to 4 (the higher the better) to the generated commit message from the three aspects: **Informativeness** (the amount of important information about the code diff reflected in the commit message), **Conciseness** (the extend of extraneous information included in the commit message), and **Expressiveness** (grammaticality and fluency). Each pair is evaluated by four volunteers, and the final score is the average of them.

To verify the agreement among the volunteers, we calculate the Krippendorff's alpha (Hayes and Krippendorff, 2007) and Kendall rank correlation coefficient (Kendall's Tau) values (Kendall, 1945). The value of Krippendorff's alpha is 0.90 and the values of pairwise Kendall's Tau range from 0.73 to 0.95, which indicates that there is a high degree of agreement between the 4 volunteers and that scores are reliable. Table 3 shows the result of human evaluation. RACE is better than other approaches in Informative, Conciseness, and Expressiveness, which means that our approach tends to generate concise and readable commit messages with more comprehensive semantics. In addition, we confirm the superiority of our approach using Wilcoxon signed-rank tests (Wilcoxon et al., 1970) for the human evaluation. Results [4] show that the improvement of RACE over other approaches is statistically significant with all p-values smaller than 0.05 at 95% confidence level.

6 Conclusion

This paper proposes a new retrieval-augmented neural commit message generation method, which treats the retrieved similar commit message as an exemplar and uses it to guide the neural network model to generate an accurate and readable commit message. Extensive experimental results demonstrate that our approach outperforms recent baselines and our framework can significantly boost the performance of four neural network models. Our data, source code and Appendix are available at https://github.com/DeepSoftwareAnalytics/RACE.

Limitations

We have identified the following main limitations:

Programming Languages. We only conduct experiments on five programming languages. Although in principle, our framework is not specifically designed for certain languages, models perform differently in different programming languages. Therefore, more experiments are needed to confirm the generality of our framework. In the future, we will extend our study to other programming languages.

Code base. Compared with purely neural network-based models, our method needs a code base to retrieve the most similar example from that. This limitation is inherited from IR-based techniques.

Training Time. In addition to modeling the information of input code diffs, our model needs to retrieve similar diffs and encode them. Thus, our model takes a long time to train (about 35 hours to train the model).

Long Code Diffs. Longer code diffs may contain more complex semantics or behaviors. Long diffs (over 512 tokens) are truncated in our approach and some information would be lost. In our future work, we will design mechanisms to better handle long diffs.

Acknowledgement

We thank reviewers for their valuable comments on this work. This research was supported

[4] Available in Appendix

by National Key R&D Program of China (No. 2017YFA0700800). We would like to thank Jiaqi Guo and Wenchao Gu for their valuable suggestions and feedback during the work discussion process. We also thank the participants of our human evaluation for their time.

References

Apache. 2011. Apache lucene.

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: an automatic metric for MT evaluation with improved correlation with human judgments. In *IEEvaluation@ACL*.

Mike Barnett, Christian Bird, João Brunet, and Shuvendu K. Lahiri. 2015. Helping developers help themselves: Automatic decomposition of code review changesets. In *ICSE (1)*, pages 134–144. IEEE Computer Society.

Raymond P. L. Buse and Westley Weimer. 2010. Automatically documenting program changes. In *ASE*, pages 33–42. ACM.

Luis Fernando Cortes-Coy, Mario Linares Vásquez, Jairo Aponte, and Denys Poshyvanyk. 2014. On automatically generating commit messages via summarization of source code changes. In *SCAM*, pages 275–284. IEEE Computer Society.

Martin Dias, Alberto Bacchelli, Georgios Gousios, Damien Cassou, and Stéphane Ducasse. 2015. Untangling fine-grained code changes. In *SANER*, pages 341–350. IEEE Computer Society.

Jinhao Dong, Yiling Lou, Qihao Zhu, Zeyu Sun, Zhilin Li, Wenjie Zhang, and Dan Hao. 2022. Fira: Fine-grained graph-based code change representation for automated commit message generation.

Lun Du, Xiaozhou Shi, Yanlin Wang, Ensheng Shi, Shi Han, and Dongmei Zhang. 2021. Is a single model enough? mucos: A multi-model ensemble learning approach for semantic code search. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2994–2998.

Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. 2013. Boa: a language and infrastructure for analyzing ultra-large-scale software repositories. In *ICSE*, pages 422–431. IEEE Computer Society.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. Codebert: A pre-trained model for programming and natural languages. In *EMNLP (Findings)*, volume EMNLP 2020 of *Findings of ACL*, pages 1536–1547. Association for Computational Linguistics.

Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *ICSE*, pages 933–944. ACM.

Andrew F Hayes and Klaus Krippendorff. 2007. Answering the call for a standard reliability measure for coding data. *Communication methods and measures*, 1(1):77–89.

Yuan Huang, Nan Jia, Hao-Jie Zhou, Xiangping Chen, Zibin Zheng, and Mingdong Tang. 2020. Learning human-written commit messages to document code changes. *J. Comput. Sci. Technol.*, 35(6):1258–1277.

Yuan Huang, Qiaoyang Zheng, Xiangping Chen, Yingfei Xiong, Zhiyong Liu, and Xiaonan Luo. 2017. Mining version control system for automatically generating commit comment. In *ESEM*, pages 414–423. IEEE Computer Society.

Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically generating commit messages from diffs using neural machine translation. In *ASE*.

Tae Hwan Jung. 2021. Commitbert: Commit message generation using pre-trained programming language model. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*, pages 26–33.

Maurice G Kendall. 1945. The treatment of ties in ranking problems. *Biometrika*, 33(3):239–251.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *NeurIPS*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*.

Qin Liu, Zihe Liu, Hongming Zhu, Hongfei Fan, Bowen Du, and Yu Qian. 2019. Generating commit messages from diffs using pointer-generator network. In *MSR*, pages 299–309. IEEE / ACM.

Shangqing Liu, Cuiyun Gao, Sen Chen, Lun Yiu Nie, and Yang Liu. 2020. ATOM: commit message generation based on abstract syntax tree and hybrid ranking. *TSE*, PP:1–1.

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 *ASE*, pages 373–384. ACM.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *ICLR*.

Pablo Loyola, Edison Marrese-Taylor, and Yutaka Matsuo. 2017. A neural architecture for generating natural language descriptions from source code changes. In *ACL (2)*, pages 287–292. Association for Computational Linguistics.

Laura Moreno, Jairo Aponte, Giriprasad Sridhara, Andrian Marcus, Lori L. Pollock, and K. Vijay-Shanker. 2013. Automatic generation of natural language summaries for java classes. In *ICPC*, pages 23–32. IEEE Computer Society.

Lun Yiu Nie, Cuiyun Gao, Zhicong Zhong, Wai Lam, Yang Liu, and Zenglin Xu. 2021. Coregen: Contextualized code representation learning for commit message generation. *Neurocomputing*, 459:97–107.

Sebastiano Panichella, Annibale Panichella, Moritz Beller, Andy Zaidman, and Harald C. Gall. 2016. The impact of test case summaries on bug fixing performance: an empirical investigation. In *ICSE*, pages 547–558. ACM.

Sheena Panthaplackel, Pengyu Nie, Milos Gligoric, Junyi Jessy Li, and Raymond J. Mooney. 2020. Learning to update natural language comments based on code changes. In *ACL*, pages 1853–1868. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *ACL*.

Jinfeng Shen, Xiaobing Sun, Bin Li, Hui Yang, and Jiajun Hu. 2016. On automatic summarization of what and why information in source code changes. In *COMPSAC*, pages 103–112. IEEE Computer Society.

Ensheng Shi, Wenchao Gub, Yanlin Wang, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2022a. Enhancing semantic code search with multimodal contrastive learning and soft data augmentation. *arXiv preprint arXiv:2204.03293*.

Ensheng Shi, Yanlin Wang, Lun Du, Junjie Chen, Shi Han, Hongyu Zhang, Dongmei Zhang, and Hongbin Sun. 2022b. On the evaluation of neural code summarization. In *ICSE*.

Ensheng Shi, Yanlin Wang, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2021a. Cast: Enhancing code summarization with hierarchical splitting and reconstruction of abstract syntax trees. In *EMNLP*.

Ensheng Shi, Yanlin Wang, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2021b. CAST: enhancing code summarization with hierarchical splitting and reconstruction of abstract syntax trees. In *EMNLP (1)*, pages 4053–4062. Association for Computational Linguistics.

Wei Tao, Yanlin Wang, Ensheng Shi, Lun Du, Shi Han, Hongyu Zhang, Dongmei Zhang, and Wenqiang Zhang. 2021. On the evaluation of commit message generation models: An experimental study. In *ICSME*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*, pages 5998–6008.

Ramakrishna Vedantam, C. Lawrence Zitnick, and Devi Parikh. 2015. Cider: Consensus-based image description evaluation. In *CVPR*.

Haoye Wang, Xin Xia, David Lo, Qiang He, Xinyu Wang, and John Grundy. 2021a. Context-aware retrieval-based deep commit message generation. *ACM Trans. Softw. Eng. Methodol.*, 30(4):56:1–56:30.

Yanlin Wang, Lun Du, Ensheng Shi, Yuxuan Hu, Shi Han, and Dongmei Zhang. 2020. Cocogum: Contextual code summarization with multi-relational gnn on umls. Technical report, Microsoft, MSR-TR-2020-16. [Online].

Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021b. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *EMNLP (1)*, pages 8696–8708. Association for Computational Linguistics.

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)*, pages 349–360. IEEE.

Frank Wilcoxon, SK Katti, and Roberta A Wilcox. 1970. Critical values and probability levels for the wilcoxon rank sum test and the wilcoxon signed rank test. *Selected tables in mathematical statistics*, 1:171–259.

Shengbin Xu, Yuan Yao, Feng Xu, Tianxiao Gu, Hanghang Tong, and Jian Lu. 2019. Commit message generation for source code changes. In *IJCAI*, pages 3975–3981. ijcai.org.

HongChien Yu, Chenyan Xiong, and Jamie Callan. 2021. Improving query representations for dense retrieval with pseudo relevance feedback. In *CIKM*, pages 3592–3596. ACM.

Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, and Xudong Liu. 2020. Retrieval-based neural source code summarization. In *ICSE*.