HybridCom: A Clone-Aware Hybrid Neural Translation and Information Retrieval Framework for Source Code Summarization

Details omitted for double-blind reviewing

***Abstract*—Automatic code comment generation is a well sought-after feature asked by developers to comprehend source code and reduce their workload in software documentation. With the recent advancements in deep learning, Neural Machine Translation (NMT) techniques have demonstrated their ability to translate among different natural languages with great success. Software engineering researchers have applied these techniques for "translating" code into natural language descriptions. However, unlike natural language corpus, code reuse is common in software development, producing many identical or similar code fragments (usually referred to as "clones") in codebases. These clones may contain highly similar code-comment pairs that are used to train and evaluate code summarization models, resulting in overly optimistic reported performance**

In this work, we review three state-of-the-art NMT-based code comment generation models and their evaluation datasets in the context of cloned code. We are the first to discover that the three benchmark datasets contain 19.4%-55.4% cloned methods in the test sets that significantly influence the reported performance of all models in terms of BLEU score. We also find that a simple information retrieval (IR) approach works better on cloned code than NMT-based models, and NMT-based models work better on non-cloned code. Thus, we propose a new clone-aware hybrid framework, namely, HybridCom, that applies an IR approach on cloned code in the test set and applies an NMT-based approach on non-cloned code. Our preliminary experimental results show that HybridCom performs better than the three state-of-the-art code summarization models by simply considering the presence of code clones.

***Index Terms*—Code summarization, Code clones, Neural machine translation, Information retrieval**

I. Introduction

Source code summarization or code-comment translation refers to the task of generating natural language descriptions from source code [1]. Code commenting is vital for software engineers to understand the meaning of source code. However, code summaries are often missing in many projects due to time constraints or lack of standardization. Thus, automatically generating natural language code comments from source code has become a well sought-after feature for many software engineering projects and developers. As a result, there has been significant effort directed towards source code summarization, with the most recent emphasis on deep learning-based approaches [2–4]. These approaches take inspiration from Neural Machine Translation (NMT) in Natural Language Processing (NLP) domain, where an encoder-decoder neural network is trained on a large text corpus and can be applied to translate text from one language to another.

Compared to the natural language corpus, code-comment translation (CCT) datasets used for the training of NMT-based code summarization models are significantly smaller [4–6]. However, it has been reported that some NMT-based code summarization models can achieve similar or even better performance than deep learning models on NLP benchmarks. For instance, the state-of-the-art transformer model achieved a 28.4 BLEU score when trained and evaluated on a corpus of 4.5M English-to-German pairs [5], while Hu et al. reported a BLEU score of 39.51 on a corpus containing 0.6M Java method-comment pairs [4]. In addition, the reported performance discrepancies are significant among existing NMT-based code summarization models even though they all collect code-comment pairs from open source repositories [2–4]. These two observations motivate us to review the properties of existing evaluation datasets for the automatic code summarization task.

Recently, Gros et al. examined the differences between CCT datasets and a popular natural machine translation (WMT) dataset. They found that a simple Information Retrieval (IR) approach can perform on par with many NMT-based models [6]. Since IR works best when similar input-output pairs (i.e., similar code-comment pairs) exist across the training and testing dataset, we hypothesize that the amount of code clones in the CCT datasets plays a vital role in the reported performance of NMT-based code summarization models because clones often have same or highly similar comments [7]. And the prevalence of cloned code-comment pairs between the training and test set in different CCT datasets may explain the two observations we mentioned earlier.

Many prior works have shown that clones are everywhere within and cross software projects [8]. However, it is unknown how the clones would affect the performance of NMT models on CCT datasets. Moreover, without understanding the performance of the existing CCT approaches in the context of clones, it is challenging for both researchers and developers to understand the true learning capability of these complex models. To fill the gap, we conduct this preliminary study first to investigate how many clones exist in the test sets of CCT datasets and then analyze the performance of NMT-based models on cloned code and non-cloned code separately.

Specifically, we analyze three state-of-the-art NMT-based code summarization models, i.e., Hybrid-DeepCom [2], FunCom [3] and Rencos [9], and their evaluation datasets in the context of clones. We find that: 1) 55.4%, 21.6%, and 19.4% of the testing methods in Hybrid-DeepCom, FunCom, and Rencos are (Type-1 and Type-2) cloned code. 2) The performances of the three models on cloned code are on average 238% higher than those on non-cloned code. 3) While the overall reported performances on the three CCT datasets vary a lot, the three models have comparative performance on non-cloned test methods. 4) A simple IR approach has a performance advantage over all NMT-based models on cloned code. 5) NMT-based models outperform the IR approach on non-cloned code. **The above findings confirm our hypothesis that the prevalence of clones in test set contributes significantly to the reported performance of code summarization models and suggest that clones and non-clones should be evaluated separately.**

Finally, we propose a new clone-aware hybrid IR NMT framework, i.e., HybridCom, for the code summarization task. HybridCom trains both an IR embedding and an NMT model during the training phase. In the test phase, HybridCom first conducts clone detection for all inputs. It then applies the IR model on cloned code and the NMT model on non-cloned code. **Our experiments on the three CCT datasets show that HybridCom can achieve an average of 18.8% increase in BLEU score over three NMT-based models.**

Overall, this work makes the following contributions.

- We analyzed three recent NMT-based code summarization models and their evaluation datasets in the context of clones. We discovered that clones greatly impact the reported model performance, and they contribute significantly to performance discrepancies among models.
- We benchmarked NMT-based models against a simple IR approach. The results indicate that the IR approach performs better on cloned code, while NMT-based approaches have the advantage on non-cloned code.
- We proposed a clone-aware hybrid framework, HybridCom, that takes advantage of both NMT and IR models. Empirical results show that the HybridCom improved the model performance of all three reviewed NMT-based models.

II. Background

A. Deep Code Comment Generation

Neural Machine Translation based source code translation approaches have recently gathered attention from many software engineering researchers. Iyer et al. proposed an LSTM-based model with an attention mechanism named CODE-NN that generates summaries for C# and SQL code snippets [10]. Hu et al. proposed both DeepCom and Hybrid-DeepCom to generate comments for Java methods using seq-to-seq models [2, 4]. Both works have extracted lexical and structural information about the code in the form of an Abstract Syntax Tree (AST) in addition to code tokens. LeClair et al. constructed a neural network model to generate Java method summary by treating two types of inputs separately: the word representation and the AST representation of code [3]. Zhang et al. proposed a technique that can retrieve both semantic and structural similar code as input for a fused encoder-decoder approach [9], which aims to address low-frequency words that are ignored in most NMT-based methods.

Gros et al. analyzed several CCT datasets and compared them with WMT19, a standard natural language dataset frequently used to train state-of-the-art natural language translators [6]. They found that the desired outputs for the CCT task are much more repetitive (e.g., comments often begin with patterns like ”returns the” and ”outputs the”), and such repetitiveness has a very strong effect on measured performance, much more so in the CCT datasets than the WMT dataset. They also reported that a naive IR approach could meet or exceed the performance of some neural models. Unlike their work, we argue that the prevalence of cloned code in CCT datasets strongly affects the measured performance of code summarization models. Moreover, we provide a new hybrid framework for code summarization by considering clones.

B. Code Clones in OSS Projects

Developers often reuse code via copy-and-paste [11]. The resulting copies are known as code clones. Lopes et al. analyzed 4.5 million GitHub projects in Java, C++, Python, and JavaScript [12]. They found that 70% of the GitHub code consists of code clones. They also found that 9%-31% of the studied projects contain 80% or more of file duplication. Gharehyazie et al. [8] analyzed cross-project code clones and found that cross-project clones account for 10%-30% of all code clones within projects. Kim et al. [7] conducted an ethnographic study to understand programmers' copy and paste programming practices and found that the text (e.g., comments) surrounding the copied code are often copied together because the cloned code often belongs to the same functionality. Therefore, identical or highly similar code-comment pairs may exist in both training and test sets due to cloning and affect the reported performance of automatic code summarization models [13].

In the literature [14], four types of clones are often considered: Type-1 clones, i.e., identical code fragments, except for differences in white-space, layout, and comments; Type-2 clones, i.e., identical code fragments, except for differences in identifier names and literal values; Type-3 clones, i.e., two copied code fragments but with modifications such as added or removed statements, and the use of different identifiers, literals, types, white spaces, layouts, and comments; Type-4 clones, i.e., semantically similar code fragments, without being syntactically similar.

III. Study Design

This preliminary study focuses on answering the following two research questions:

RQ1: How are code clones distributed in CCT datasets, and how the state-of-the-art NMT-based and IR models perform on cloned and non-cloned code?

RQ2: Can a clone-aware IR NMT hybrid framework improve the performance of the state-of-the-art NMT-based code comment generation models?

A. Datasets

To answer the two RQs, we consider three recent NMT-based CCT models, namely Hybrid-DeepCom, FunCom, Rencos, and their datasets. We briefly describe the datasets as below:

Hybrid-DeepCom Hu et al. [4] collected this dataset from 9,714 open source Java Github projects. Only methods with Javadoc descriptions are extracted. The author roughly follows a 90-5-5 split by method, i.e., 445,912 (90%) method-comment pairs for training, 20,00 pairs for validation, and 20,000 pairs for test.

FunCom LeClair et al. [3] started with the Sourcerer dataset [15], which contains 51 million Java methods from 50,000 projects. The authors removed code snippets with auto-generated comments by removing any methods from files that include phrases such as "generated by". The final dataset contains 2.1 million code-comment pairs. The authors then randomly split the dataset by project, i.e., 90% of projects are used for training, 5% for validation, and 5% for test.

Rencos Zhang et al. [9] started with the code-comment pairs extracted by Hu et al. [16] from 9,714 popular Java projects on Github. They split the dataset by method, following a 80-10-10 split. The authors mentioned that they removed duplicated samples from the testing set, but it is unclear how they identify those duplicated samples.

To summarize, the three CCT datasets are split into training, validation, and test sets either by project or by method. Although the creators of Rencos mentioned that they remove duplicated samples from the test set, a detailed process is not provided. None of the datasets' creators reported any statistics of clones.

B. Baselines and Experiment Setup

For the three NMT-based models, i.e., Hybrid-DeepCom, FunCom, and Rencos, we downloaded the source code provided by the authors and kept their split of training, validation, and test sets.

We implement a simple IR approach, named BOW-IR, to benchmark against the NMT-based models. BOW-IR takes all code snippets in a CCT dataset as input and represents them as vectors in the form of "bag-of-words" (BOW). Then, for each test method, the nearest neighbor, according to the Euclidean distance, is retrieved from the training set. Finally, the comment of the nearest neighbor is returned as the generated comment of the test method. Unlike the NMT-based models, BOW-IR is more efficient and costs less computation as it does not need to construct and tune a complex deep model.

C. Evaluation Metric

To align with Hu et al. [2], we use the BLEU-4 score [17] to evaluate the performance of all models. The BLEU score is widely used to assess the quality of machine translation systems. The score measures the similarity between the actual comment and the generated comment by first calculating the modified n-gram (for BLEU-4, n=1,2,3,4) precisions of a generated comment to the actual comment, then measuring the average modified n-gram precision with a penalty for overly short sentences.

D. Clone Detection

In this preliminary study, we focus on Type-1 and Type-2 clones across training and test sets (referred to as clones in later sections). We choose Type-1 and Type-2 clones because they are more likely to have similar comments and can be efficiently and accurately detected by existing clone detection tools [18]. Clone pairs may exist within training/validation/test sets or across any two of the three sets. We choose to focus on across training and test clones because such cloned test methods may be much easier to summarize and thus overestimate the performance of a model compared to the performance that actual users of the model observed on non-cloned code.

To answer RQ1 and identify clones, we run SourcerCC [18], a common token-based clone detection tool that was designed for large-scale clone detection, on all three CCT datasets. SourcerCC uses a "Overlap" (can be replaced by Jaccard similarity) metric to measure the overlap between code pairs and identify clone pairs. SourcerCC can reliably detect Type-1 and Type-2 clones, with a reported 91% precision and 100% recall. We use the suggested threshold, i.e., 0.8, to identify Type-2 clones.

E. HybridCom Framework

We propose HybridCom as a clone-aware framework (as shown in Figure 1) where the cloned methods in the test set are summarized using a simple IR model, and the remaining non-cloned methods are summarized using an NMT-based model.

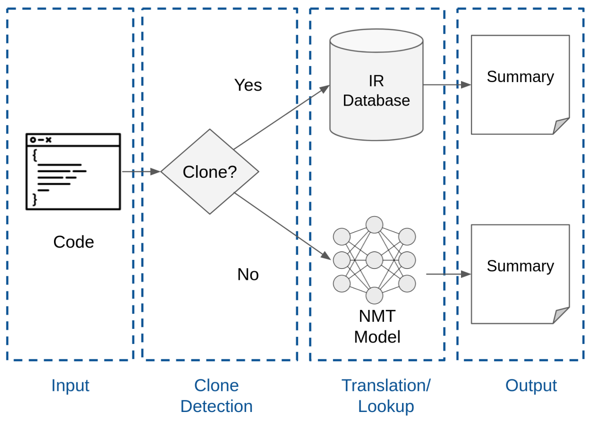

Fig. 1: HybridCom Framework

HybridCom does not modify the existing NMT models and their training process. It simply adds a clone identification process on top of an NMT-based model in the test phase. During training, HybridCom collects training data and encodes each training method using the BOW representation. The encoded methods and their comments serve as a retrieval base. HybridCom also assumes that there was an NMT-based CCT model trained on all training data in the training phase. During testing, given a test method, HybridCom first converts it into its BOW format and then searches for its nearest neighbor in the retrieval base using Jaccard similarity. If the Jaccard similarity between the nearest neighbor and the test method is greater than 0.8, HybridCom will identify the test method as cloned and directly output the comment of the nearest neighbor as the generated comment. Otherwise, the test method will be identified as non-cloned code, and the trained NMT-based model will be applied to generate the comment.

IV. Results and Discussion

A. RQ1. Prevalence of Code Clones in CCT Datasets and Their Impact

Following the study design in Section III, we reuse the original training/validation/test split on three CCT datasets and then run SourcerCC to identify cloned methods and non-cloned methods in the three test sets. We summarize the statistics of the identified clones in Table I.

TABLE I: Number of Methods and Clones in Three CCT Datasets

	Hybrid-DeepCom	FunCom	Rencos
# Training	445,812	1,954,807	69,708
# Validation	20,000	104,273	8,714
# Test	20,000	90,908	6,489
Clones(% of test)	11,088 (55.4%)	19,621 (21.6%)	1,262 (19.4%)

Results in Table I show that a significant number of clones exist in all CCT datasets. FunCom and Rencos have 21.6% and 19.4% clones in their test sets, respectively. Hybrid-DeepCom contains a significantly higher percentage of clones, i.e., 55.4%. We postulate this could be due to several reasons. First, Hybrid-DeepCom's split is by method, and therefore many within-project clone pairs may distribute on both training and test sets. Second, neither the code with auto-generated comments (considered in FunCom) nor duplicate samples (considered in Rencos) are removed when creating Hybrid-DeepCom.

To investigate the impact of clones on the reported performance of NMT-based CCT models, we train and test the three selected models on their own evaluation datasets and recalculate their performance on cloned and non-cloned methods. The results are reported in Figure 2. **On average, the performance of three NMT-based models on clones are 38% higher than those on non-clones in terms of BLEU score.** As a result, the overall reported BLEU scores are all inflated due to the prevalence of clones across training and test sets. We also observe that although the overall performances of the three models vary a lot, their performances on non-cloned code are much more comparable. For instance, the maximum overall performance difference among the three models is 18.68 (39.20-20.52), while the value is 2.12 (16.90-14.78) if we only consider non-cloned code. This finding also indicates that the prevalence of clones may explain the large reported performance discrepancies of state-of-the-art NMT-based models on different CCT datasets.

Figure 3 shows the performance of the simple BOW-IR model on three datasets in the context of cloned code. We

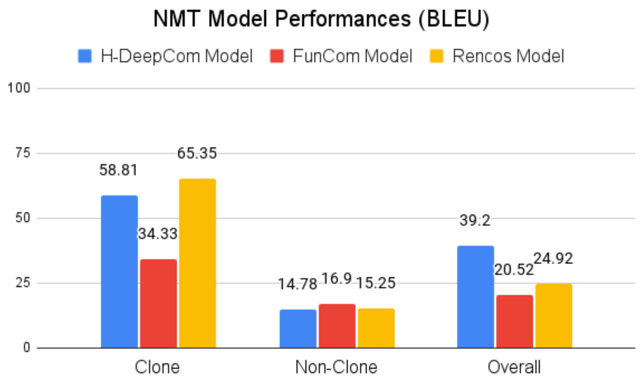

Fig. 2: NMT Model Performance on each Dataset

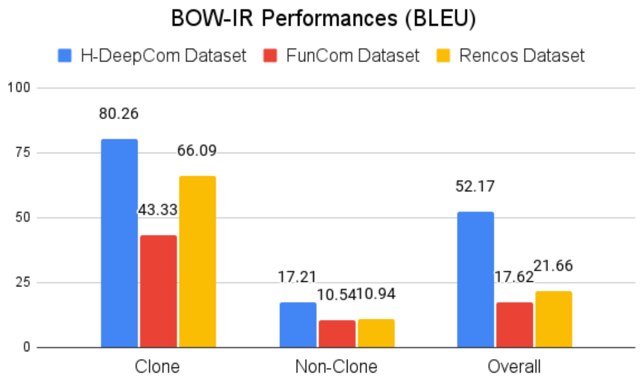

Fig. 3: BOW-IR Performance on each Dataset

can observe that BOW-IR performs worse than FunCom and Rencos and better than Hybrid-DeepCom if we consider the overall reported BLEU scores. If we inspect its performance on clones and non-clones separately, we find that **BOW-IR can achieve much better results on cloned code compared with the state-of-the-art NMT models.** On the other hand, NMT-based models are generally better than the BOW-IR model when applied to non-clones. The only exception happens on the Hybrid-DeepCom dataset, where BOW-IR model can achieve the best performance on non-cloned code. A possible reason is that there might be highly similar code-comment pairs in the Hybrid-DeepCom dataset not captured by the code clone detection tool.

B. RQ2. HybridCom Vs. NMT-based Models

Results in RQ1 show the advantages of using IR for summarizing cloned codes and using NMT models for summarizing non-cloned code. This finding motivates our clone-aware HybridCom framework, which takes advantage of IR and NMT-based models. For each dataset, we run our HybridCom framework and the overall performance of HybridCom is shown in Figure 4. We can observe that HybridCom shows improvements in overall performance for all three NMT models. On average, the overall performance is increased by 18.8% compared to only using NMT-based models. On the Hybrid-DeepCom dataset, which contains significantly more clones,

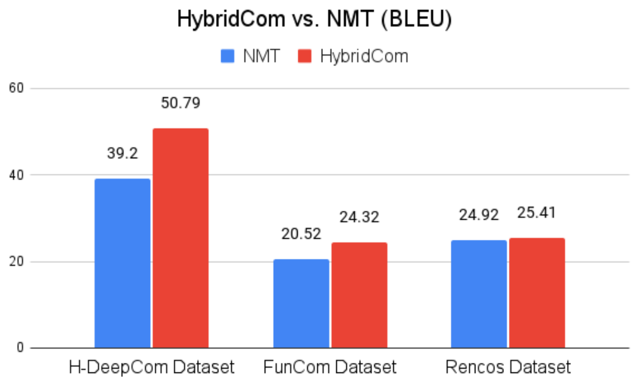

Fig. 4: HybridCom and NMT Performances Comparison

HybridCom can achieve a much better BLEU score, which mainly due to the fact the IR performs much better on clones.

Hybrid-DeepCom takes extra time in identifying clones compared to NMT-based models. However, the time added is only a few milliseconds for each test code. This extra time can be easily offset by using IR instead of an NMT model when decoding cloned code.

V. Conclusion and Future Directions

In this work, we revisit the evaluation of three state-of-the-art NMT-based code summarization models in the context of clones. We find that code clones make a large portion of the existing CCT datasets and significantly impact their reported performance. We also find that a simple IR approach outperforms NMT-based models on clones, while NMT-based models are generally better for non-clones. We propose HybridCom, a clone-aware framework that can take advantage of both IR and NMT-based models for the code summarization task. Empirical results show that HybridCom can improve the overall performance of all three NMT models by a material margin.

Our preliminary study demonstrates the importance of considering clones in designing and evaluating models for the code summarization task. We suggest that researchers should consider the nature of clones in open-source repositories and examine the performance of models on clones and non-clones separately.

In the future, we would like to conduct a qualitative analysis of the comments generated by IR and NMT-based models for both clones and non-clones. Such a qualitative study would provide in-depth knowledge of the naturalness of the generated summaries that a quantitative measurement like the BLEU score can not capture. We would also like to examine the impact of other clones within and across train/validation/test sets on the performance of CCT models.

References

[1] Y. Zhu and M. Pan, "Automatic code summarization: A systematic literature review," *CoRR*, vol. abs/1909.04352, 2019.

[2] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *Proceedings of the 26th Conference on Program Comprehension, ICPC*. ACM, 2018, pp. 200–210.

[3] A. LeClair, S. Jiang, and C. McMillan, "A neural model for generating natural language summaries of program subroutines," in *Proceedings of the 41st International Conference on Software Engineering, ICSE*, J. M. Atlee, T. Bultan, and J. Whittle, Eds. IEEE / ACM, 2019, pp. 795–806.

[4] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation with hybrid lexical and syntactical information," *Empirical Software Engineering*, vol. 25, no. 3, pp. 2179–2217, 2020.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, 2017, pp. 5998–6008.

[6] D. Gros, H. Sezhiyan, P. Devanbu, and Z. Yu, "Code to comment "translation": Data, metrics, baselining & evaluation," in *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*. IEEE, 2020, pp. 746–757.

[7] M. Kim, L. Bergman, T. Lau, and D. Notkin, "An ethnographic study of copy and paste programming practices in oopl," in *Proceedings. 2004 International Symposium on Empirical Software Engineering, 2004. ISESE'04*. IEEE, 2004, pp. 83–92.

[8] M. Gharehyazie, B. Ray, M. Keshani, M. S. Zavosht, A. Heydarnoori, and V. Filkov, "Cross-project code clones in github," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1538–1573, 2019.

[9] J. Zhang, X. Wang, H. Zhang, H. Sun, and X. Liu, "Retrieval-based neural source code summarization," in *ICSE '20: 42nd International Conference on Software Engineering,*, G. Rothermel and D. Bae, Eds. ACM, 2020, pp. 1385–1397.

[10] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Summarizing source code using a neural attention model," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016*. The Association for Computer Linguistics, 2016.

[11] Y. Dang, D. Zhang, S. Ge, C. Chu, Y. Qiu, and T. Xie, "Xiao: Tuning code clones at hands of engineers in practice," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 369–378.

[12] C. V. Lopes, P. Maj, P. Martins, V. Saini, D. Yang, J. Zitny, H. Sajnani, and J. Vitek, "Déjàvu: a map of code duplicates on github," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 1–28, 2017.

[13] M. Allamanis, "The adverse effects of code duplication in machine learning models of code," in *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, H. Masuhara and T. Petricek, Eds. ACM, 2019, pp. 143–153.

[14] A. Sheneamer and J. Kalita, "A survey of software clone detection techniques," *International Journal of Computer Applications*, vol. 137, no. 10, pp. 1–21, 2016.

[15] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes, "Sourcerer: a search engine for open source code supporting structure-based search," in *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, 2006, pp. 681–682.

[16] X. Hu, G. Li, X. Xia, D. Lo, S. Lu, and Z. Jin, "Summarizing source code with transferred api knowledge," 2018.

[17] K. Papineni, S. Roukos, T. Ward, and W. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. ACL, 2002, pp. 311–318.

[18] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "Sourcerercc: scaling code clone detection to big-code," in *Proceedings of the 38th International Conference on Software Engineering, ICSE*. ACM, 2016, pp. 1157–1168.