Correlating Automated and Human Evaluation of Code Documentation Generation Quality

XING HU, School of Software Technology, Zhejiang University, China
QIUYUAN CHEN and HAOYE WANG, College of Computer Science and Technology, Zhejiang University, China
XIN XIA, Faculty of Information Technology, Monash University, Australia
DAVID LO, School of Information Systems, Singapore Management University, Singapore
THOMAS ZIMMERMANN, Microsoft Research, USA

Automatic code documentation generation has been a crucial task in the field of software engineering. It not only relieves developers from writing code documentation but also helps them to understand programs better. Specifically, deep-learning-based techniques that leverage large-scale source code corpora have been widely used in code documentation generation. These works tend to use automatic metrics (such as BLEU, METEOR, ROUGE, CIDEr, and SPICE) to evaluate different models. These metrics compare generated documentation to reference texts by measuring the overlapping words. Unfortunately, there is no evidence demonstrating the correlation between these metrics and human judgment. We conduct experiments on two popular code documentation generation tasks, code comment generation and commit message generation, to investigate the presence or absence of correlations between these metrics and human judgments. For each task, we replicate three state-of-the-art approaches and the generated documentation is evaluated automatically in terms of BLEU, METEOR, ROUGE-L, CIDEr, and SPICE. We also ask 24 participants to rate the generated documentation considering three aspects (i.e., language, content, and effectiveness). Each participant is given Java methods or commit diffs along with the target documentation to be rated. The results show that the ranking of generated documentation from automatic metrics is different from that evaluated by human annotators. Thus, these automatic metrics are not reliable enough to replace human evaluation for code documentation generation tasks. In addition, METEOR shows the strongest correlation (with moderate Pearson correlation r about 0.7) to human evaluation metrics. However, it is still much lower than the correlation observed between different annotators (with a high Pearson correlation r about 0.8) and correlations that are reported in the literature for other tasks (e.g., Neural Machine Translation [39]). Our study points to the need to develop

This research was partially supported by the National Science Foundation of China (No. U20A20173) and the National Research Foundation, Singapore under its **Industry Alignment Fund—Pre-positioning** (**IAF-PP**) Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

Authors' addresses: X. Hu, School of Software Technology, Zhejiang University, No. 1689 Jiangnan Road, Ningbo, Zhejiang, 315048, China; email: xinghu@zju.edu.cn; Q. Chen and H. Wang, College of Computer Science and Technology, Zhejiang University, Road 38 West Lake District, Hangzhou, Zhejiang, 310027, China; emails: {chenqiuyuan, why_}@zju.edu.cn; X. Xia (corresponding author), Faculty of Information Technology, Building 6, 29 Ancora Imparo Way, Clayton Campus, Monash University VIC 3800; email: xin.xia@acm.org; D. Lo, School of Information Systems, Singapore Management University, 80 Stamford Road, Singapore 178902; email: davidlo@smu.edu.sg; T. Zimmermann, Microsoft Research, 1 Microsoft Way, Redmond, WA 98052; email: tzimmer@microsoft.com.

1049-331X/2022/07-ART63 $15.00
https://doi.org/10.1145/3502853

specialized automated evaluation metrics that can correlate more closely to human evaluation metrics for code generation tasks.

CCS Concepts: • **Software and its engineering → Software creation and management**; **Documentation;**

Additional Key Words and Phrases: Code documentation generation, evaluation metrics, empirical study

ACM Reference format:
Xing Hu, Qiuyuan Chen, Haoye Wang, Xin Xia, David Lo, and Thomas Zimmermann. 2022. Correlating Automated and Human Evaluation of Code Documentation Generation Quality. *ACM Trans. Softw. Eng. Methodol.* 31, 4, Article 63 (July 2022), 28 pages.
https://doi.org/10.1145/3502853

1 INTRODUCTION

During software maintenance and development, program comprehension is the main activity for developers [51]. High-quality documentation such as code comments, commit messages, and release notes can help developers better understand programs [13]. Unfortunately, due to tight project schedules and other reasons, documentation is often missing, or incomplete. Therefore, many techniques are proposed to generate the documentation automatically. These techniques relieve developers from writing documentation and help them understand existing software.

Earlier works usually exploit manually-crafted templates [36] and Information Retrieval techniques [16, 17] to assemble key terms into the documentation. These techniques usually rely on heuristics and stereotypes to select the information that should be included in the documentation. Then, they evaluate the generated documentation through human evaluation in terms of expressiveness (readable and understandable), content adequacy (important information about the class reflected in the documentation), and conciseness (extraneous information in documentation) [36]. However, the scale of these evaluations tends to be small—usually no more than a few hundred sentences examined by a small number of raters. Thus, it can be difficult to draw firm conclusions about the overall quality of the generated documentation.

In recent years, there is an emerging interest in building deep learning models to generate code documentation. These techniques take advantage of neural networks and the large available open source code repositories [20, 50] to capture lexical and syntactical features from source code. At the same time, various automatic metrics such as BLEU [39], ROUGE [27], METEOR [4], CIDEr [48], and SPICE [2] in the **Natural Language Processing** (**NLP**) domain are adopted to evaluate code documentation models. Generally, these metrics measure different models by comparing overlapping text between the reference and the generated text. The model can achieve higher scores if there are more overlapping words.

Some works evaluate the quality of models through automatic metrics accompanied by a human study, in which programmers are asked to rate various aspects of the generated contents or naturalness of the documentation. For example, Jiang et al. [23] and Hu et al. [20] asked developers to give scores by comparing the semantic similarity between the generated documentation and the reference text. Wei et al. [50] asked developers to score different documentation from three aspects: the similarity of generated comments and references, naturalness (grammatical correctness and fluency of the generated comments), and informativeness (the amount of content carried over from the input code to the generated comments, ignoring fluency of the text). Because doing user studies is time-consuming, costly, and relying on subjective judgments, some works only compare different models through automatic metrics [19, 49] without human evaluation.

Since practical considerations have forced the field to rely on automated metrics, it is crucial to determine how well these metrics compare to human judgments. A reliable automatic metric can

serve as a proxy for human evaluation which is considerably more expensive and time-consuming. Judging whether, and to what extent, automatic metrics concur with the human evaluation has not been sufficiently established in existing studies.

To figure out whether automated metrics are reliable and can indeed replace human judgment in the domain of automatic code documentation generation, we explore the correlation between five automatic metrics and six human evaluation metrics for code documentation generation tasks. The automatic metrics used in this article are BLEU, METEOR, ROUGE-L, CIDEr, and SPICE which are widely used in various documentation generation works [19, 20, 23, 32, 49]. We investigate the human evaluation metrics used in previous studies and select six widely used metrics. These metrics are used to evaluate models from three aspects, including *Language-related* (measures natural language features and ignore the documentation's contents), *Content-related* (measures the amount of contents carried from the input code to the generated documentation), and *Effectiveness-related* (evaluate whether generated documentation is useful or helps developers understand programs). Each aspect contains two metrics: Naturalness and Expressiveness for the *Language-related* aspect; Content Adequacy and Conciseness for the *Content-related* aspect; Usefulness (evaluate how useful the documentation is) and Code Understandability (evaluate to what extent the generated documentation can help developers understand programs) for the *Effectiveness-related* aspect. In this article, we conduct experiments on two documentation generation tasks, the code comment generation task and the commit message generation task. For each task, we first replicate three state-of-the-art approaches (i.e., Hybrid-DeepCom [20], Code2Seq [1], and Re^{2}Com [50] for code comment generation task; NMT [23], NNGen [32], and PtrGNCMsg [30] for commit message generation task). Then, we evaluate them by using automatic metrics and human evaluation metrics. We recruit 24 evaluators to score 200 randomly sampled comments and commit messages, respectively. Then, we analyze the correlation between different automatic and human evaluation metrics.

Our study aims at answering the following research questions:

RQ1: What are the results of state-of-the-art approaches on automatic metrics and human evaluation metrics?
We investigate this RQ to compare the generated documentation from automatic metrics and human evaluation metrics. The automatic metrics mainly evaluate generated documentation by counting the number of overlapping N-grams between it and human-written reference text. The human evaluation metrics are computed based on user study participant feedback; each participant is asked to give a score for each documentation with respect to a given source code/*diff*.

RQ2: What are the correlation inside human evaluation metrics and automatic evaluation metrics, respectively?
From the experiments in RQ1, we present the overall scores of documentation generated by different approaches. Another important question is whether automatic metrics or human evaluation metrics are consistent in evaluating the generated documentation. We measure the Kendall τ correlation and Pearson r correlation to explore whether these metrics are concordant while scoring different approaches.

RQ3: Do automatic metrics such as BLEU, METEOR, ROUGE-L, CIDEr, and SPICE correlate with human judgment on generated documentation?
In neural machine translation literature, automatic metrics have been shown to correlate well with human judgment and can replace human raters [9]. In this RQ, we propose to establish the correlation between automatic metrics and human evaluation metrics. According to the correlation, we can quantify the extent of automatic metrics reflecting the human perspectives and find the most relevant automatic metrics to human judgments. We follow Coughlin et al. [9] and

compute the Pearson's correlation r and Kendall's correlation τ between automatic and human evaluation metrics to explore whether automatic metrics can reflect human judgments on generated documentation.

Our interpretation of τ and r is based on Hinkle et al.'s scheme [18]: negligible correlation ($|\tau/r| < 0.3$), low correlation ($0.3 \leq |\tau/r| < 0.5$), moderate correlation ($0.5 \leq |\tau/r| < 0.7$), high correlation ($0.7 \leq |\tau/r| < 0.9$), and very high correlation ($0.9 \leq |\tau/r| \leq 1$).

The contributions of our work are shown as follows:

- We replicate and evaluate state-of-the-art approaches on two documentation generation tasks, comment generation and commit message generation. To evaluate the generated documentation, we conduct a survey that asks 24 annotators to give Likert scores from six aspects. In addition, we also calculate automatic metrics including BLEU, METEOR, ROUGE-L, CIDEr, and SPICE.
- We investigate the correlation inside automatic metrics and human evaluation metrics. We find high correlation between *Content-Related* and *Effectiveness-Related* metrics.
- We present the Pearson the correlation between automatic metrics and human evaluation metrics. METEOR is more correlated (with moderate Pearson correlation r about 0.7 and Kendall correlation τ about 0.5) to human evaluation metrics than other automatic metrics. Although it achieves a relatively high correlation, it is weaker when compared to correlation between different annotators (Pearson correlation r about 0.8 and Kendall correlation τ about 0.7).

The remainder of this article is organized as follows. In Section 2, we provide background knowledge of automatic documentation generation and evaluation metrics. In Section 3, we introduce the methodology to conduct the experiments and survey. In Sections 4 and 5, we introduce the details of state-of-the-art models and evaluation metrics we used in this article. Then, we present answers the answers to the three research questions. We give the discussion and threats to validity in Sections 6 and 7. Finally, we conclude the whole study and summarise future work in Section 8.

2 BACKGROUND

2.1 Automatic Documentation Generation

Code documentation is the essential information for developers during the program comprehension. It exists in all phases of the software development cycle, e.g., comments, commit messages, and release notes. For instance, code comments are used to describe the functionality of programs. Commit messages are helpful for developers to understand the software evolution. However, much code documentation is incomplete [26], which costs time and efforts when understanding the software. Therefore, automatic documentation generation is a crucial task to help developers understand programs. This task aims at giving natural language descriptions for the source code or other software artifacts. Generally, these generated descriptions should reflect programs' intent. Recent studies [19, 22, 50] formulate this task as a translation process that translates programming language into natural language. Hence, evaluation metrics used in NMT are exploited in documentation generation. In this article, we mainly explore whether these metrics can reflect human perspectives on code documentation.

2.2 Commit, *diff*, and Commit Messages

The commit message generation task aims at generating a commit message according to the *diff* of a change. When developers submit a code change to the version control system like Git, they can enclose a message to describe the change and/or the reason for the change. The changes can be represented by *diff* which can be generated by the *git diff* command in Git. The commit messages help

understand the purpose of the code changes and even the evolution of software. Unfortunately, developers often submit low quality or even empty commit messages [11]. Recently, several methods have been proposed to generate commit messages for code changes automatically.

2.3 Evaluation Metrics

The evaluation metrics used in the documentation generation task mainly consists of two categories: automatic metrics and human evaluation metrics. Automatic metrics such as BLEU [39], METEOR [4], and ROUGE-L [27] are widely used in many NLP tasks, e.g., machine translation [47] and text summarization [40]. These metrics are directly used to measure code documentation generation approaches when machine translation models are exploited to solve this task. They are designed to measure the semantic similarity between the generated documentation and references considering the overlapping n-grams between them.

Human evaluation metrics generally ask human annotators to score generated documentation from different aspects. Some works ask human participants to compare the semantic similarity between the generated documentation and the reference (i.e., human-written documentation) [20, 23]. If the two sequences are semantic similar, the generated documentation will achieve a high score. In this case, the human annotators evaluate the generated documentation without the source code. Hence, human annotators can not judge whether the generated documentation describes the source code accurately and whether it helps understand the programs without the source code.

Other works ask human annotators to judge the documentation from various aspects such as naturalness and content given the input source code. Moreno et al. [36] asked 22 programmers to judge the content adequacy, conciseness, and expressiveness of automatically generated summaries for 40 classes. Similarly, Mcburney et al. [35] asked participants to answer questions, including a summary's accuracy, content adequacy, and conciseness. Liu et al. [32] ask participants to score the semantic similarities between generated commit messages and references given the commit diffs. Wei et al. [50] evaluated different techniques from three metrics, naturalness, informative, and similarity. These human evaluation results can illustrate developers' perspectives on the generated documentation. Recently, Stapleton et al. [45] asked human annotators to complete the code comprehension task and code writing task given the comment generated by Leclair et al. [25]. Their results show that the BLEU and ROUGE are uncorrelated to code comprehension. However, their work is only tied to a particular model for code comment generation tasks and can not judge the models' ranking. Besides, they do not give a correlation between these automatic metrics and human evaluation metrics on models' evaluation.

In this article, we explore whether automatic metrics and human evaluation metrics are consistent with ranking different approaches to documentation generation. In addition, we also give the correlation between them that helps researchers to better understand and utilize them.

2.4 Motivating Examples

Figure 1 illustrates an example in which the generated comment (by Re^{2}Com [50]) has high automatic scores (ROUGE-L: 65.87%) because the overlapping N-gram "attempts to transition the entry from" contributes positively to the automatic metric scores. However, we ask six developers how useful the reference text and the generated comment are with respect to the source code. We find that all of them think the usefulness of the generated comment is minimal, whereas the reference text is much more useful.

Another example is shown in Figure 2 in which the generated commit message (by NNGen [32]) has low automatic scores (BLEU: 7%) since there are few overlapping words between it and the reference text. However, human annotators think it is very helpful for the *diff*'s understandability because it succinctly explains the change of the import statement clearly.

Source Code:

```
boolean goFromRetiredToDead(){
    return compareAndSet(Status.RETIRED,Status.DEAD);
}
```

Reference: attempts to transition the entry from retired to dead when releasing the handle.

Generated comments by Re^2Com: attempts to transition the entry from idle to customise when evicting from the idle cache.

Fig. 1. An example of generated comment with high automatic scores. *Reference* is the human-written documentation and *Generated comments by Re^{2}Com* is the machine-generated documentation by Re^{2}Com [50].

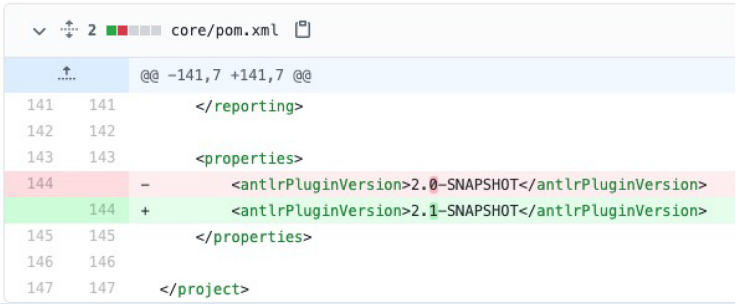


Fig. 2. An example of generated commit message with low automatic scores.

We can observe that for this example there is a huge gap between automatic evaluation results and human perspectives. Therefore, the correlation between automatic metrics and human evaluation metrics for code documentation generation tasks require further exploration.

3 METHODOLOGY

In this section, we describe our methodology for evaluating the quality of the generated documentation. It mainly contains two phases, i.e., model replication and survey. In phase 1, we replicate three state-of-the-art techniques for each task (comment generation and commit message generation) and evaluate them using automatic metrics. After obtaining the generated documentation, we use a survey that asks participants to score each generated documentation with respect to a piece of code/diff. In this study, we conducted an IRB-approved human study involving 24 evaluators that have more than three years of studying/working experience in the software development process to complete human evaluation. Among the 24 evaluators, 15 evaluators are professional developers and nine evaluators are graduate students in computer science. Each participant is asked to complete surveys for the two tasks. Each case is scored by six annotators and we use the average score of the six annotators as the final score for each case. In the following subsections, we present the details of our experiments and survey.

3.1 Experiments Details

In this section, we replicate state-of-the-art models by re-running the source code provided by the authors of the code documentation generation tools. It mainly includes three steps, i.e., dataset selection, model training, and automatic evaluation.

3.1.1 Dataset Selection. For the code comment generation task, we use the cross-project dataset provided by Hu et al. [20]. It consists of more than 8,000 projects in the training set and 971 projects in the test set. After processing, it consists of 455 k and 5 k <Java method, comment> pairs in training and test sets, respectively.

For commit message generation task, we use the dataset provided by Liu et al. [32]. This dataset is derived from Jiang et al. [23]. Liu et al. [32] removed trivial messages from the original dataset. It consists 26 K, 3 k, and 3 k commits in training, validation, and test sets, respectively.

3.1.2 Model Training. For the code comment generation tasks, we replicate three state-of-the-art techniques, including Hybrid-DeepCom, Code2Seq, and Re^{2}Com. We follow the source code provided by the authors and use the same parameters. We retrain these three models and get the generated documentation.

In particular, the Code2Seq model is proposed to conduct "extreme code summarization" that generates a method name instead of a sentence of natural language (i.e., comment). Therefore, we modify the corresponding part of the model (i.e., the generator module of method names) to enable it to generate code comments instead of method names (Code2Seq is often used in generating code comments in literature [7].

For the commit message generation task, we replicate another three models, i.e., NMT, NNGen, and PtrGNCMsg. As Liu et al. [32] provide the generated commit messages of NMT and NNGen, we reuse the given results. Then, we retrain the PtrGNCMsg model with the same dataset as the other two models.

We conduct all experiments on a Linux server with the NVIDIA Tesla T4 GPU with 16 GB memory. During the training, we keep the original hyperparameters provided by the authors (e.g., model structures and training strategies) to reproduce their experiments.

3.1.3 Automatic Evaluation. After obtaining the generated documentation, we compute the overall scores of BLEU, METEOR, ROUGE, CIDEr, and SPICE by the tool NLG [42]. To get the correlation between the human evaluation and automatic evaluation, we need to compare scores for each generated documentation. Thus, we compute BLEU, METEOR, ROUGE, CIDEr, and SPICE scores for each documentation.

3.2 Survey

3.2.1 Survey Design. Our human evaluation survey aims at scoring each documentation. According to the previous studies, we exploit human evaluation metrics from three aspects, including *Language-related*, *Content-related*, and *Effectiveness-related*. The detailed descriptions of these metrics are shown in Section 4.

Participants are shown a task description and the criteria details at the start of the survey. Participants should give scores (on a scale between 1 and 5) for each documentation by reading the given cases. The criteria details are described in Table 2.

Then, we present two examples with the recommended scores and show the explanation for recommended scores. We present an example that consists the documentation given a specific Java method in Figure 3. For each case, participants could view the Java method/*diff*, along with four pieces of documentation. One piece of documentation is the ground truth reference and the other three pieces of documentation are generated by different techniques. Then, participants are asked

Java Method

```
public FieldRef(String declClass, String fieldType,
                String fieldName){
    mDeclClass = declClass;
    mFieldType = fieldType;
    mFieldName = fieldName;
}
```

Documentation

Doc 1: initializes a new field reference.
Doc 2: creates a primitive type
Doc 3: perform a http delete request and returns the result if it is already
Doc 4: constructs an analysiscontext with a given field name field type and conversion.

Recommended Scores & Explanation for Doc 4:
Naturalness: 4 (Has a grammatical error, namely, “field name field type” (should be “field name, field type”).)
Expressiveness: 4 (Can understand the meaning of this sentence.)
Content Adequacy: 2 (Contains little important information, including “field name” and “field type”.)
Conciseness: 3 (Has much unnecessary information, for example “analysiscontext”.)
Usefulness: 2 (Is useless.)
Code Understandability: 2 (Is unhelpful to understand this method.)

Fig. 3. An Example with recommended scores we gave to the participants in the survey study.

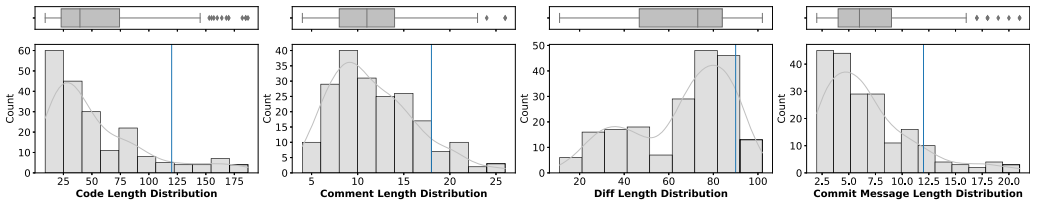


Fig. 4. Length distribution of the selected samples. Blue lines represent 90% of cases’ length. Figure 4(a) and (b) are distributions of code lengths and comment lengths for the comment generation task; Figure 4(c) and (d) are distributions of diff lengths and commit message lengths for the commit message generation tasks.

to score the documentation by reading the given Java methods/*diffs* and their corresponding documentation. Note that the participants do not know who/what generated the documentation. They give scores by reading and understanding Java methods or diffs, and compare their understanding with the generated documentations.

Participants are allowed to search the Internet for related information and unfamiliar concepts.

3.2.2 Cases Selection. We randomly select 200 cases and four corresponding documentation for each task, in which one documentation is the ground truth reference text and the other three are generated by different approaches. The length distribution of selected cases is shown in Figure 4. The average lengths of code, comment, diff, and commit messages are 54.41, 11.61, 65.81, and 6.84, respectively. For the comment generation task, most cases (90%) have less than 120 tokens in Java methods and 18 words in comments. For the commit message task, most cases (90%) have less than 90 tokens in diffs and 12 words in commit messages.

3.2.3 Participant Selection. In this study, we invited 24 participants with a combination of graduate computer science students and professional industrial developers.

3.2.4 Survey Procedure. We invite 24 evaluators to participate in our user study; all of these participants have more than three years of studying/working experience in the software development process and are familiar with Java programming language. Each annotator scores 50 cases for the comment generation task and 50 cases for the commit message generation task. Each case is rated by six annotators and participants are asked to complete the survey independently. Then, we compute the average score for each case. We do not limit the amount of time for evaluators to complete the user study.

We recommend that participants rest for at least half an hour for every half hour annotation. On average, participants cost 1.35 minutes to give their rating for one case.

Table 1. A Summary of the Evaluation Metrics Considered in this Study

Metric	Proposed to evaluate	Underlying idea
BLEU [39]	Machine translation	n-gram precision
METEOR [4]	Machine translation	n-gram with synonym matching
ROUGE-L [27]	Document Summarization	n-gram recall
CIDEr [48]	Image description generation	tf-idf weighted n-gram similarity
SPICE [2]	Image description generation	Scene-graph synonym matching

3.3 Replication Package

All the data and source code used in our study is publicly available.[1]

4 EVALUATION METRICS

In this section, we introduce the evaluation metrics used in this article, including three automatic metrics and six human evaluation metrics from three aspects.

4.1 Automatic Metrics

In this article, we select five automatic metrics (i.e., BLEU, METEOR, ROUGE-L, CIDEr, and SPICE) that are often used in the automatic evaluation of software documentation generation tasks. A summary of metrics investigated in our study is given in Table 1. The scores of BLEU, METEOR, ROUGE-L, and SPICE are in the range of [0,1] and usually reported in percentages. The higher the score, the closer the generated documentation is to the reference. If the generated documentation is completely equal to the reference, these scores become 100%. But CIDER is not between 0 and 1, and thus it is reported in real values. These scores range from 1 to 100 as a percentage value. All automatic metrics are computed by scripts provided by *pycocoevalcap*.[2]

4.1.1 BLEU. BLEU [39] is one of the most common metrics used to evaluate machine translation tasks. It is usually used to measure the textual similarity between candidate hypotheses and the reference. Generally, it calculates the modified n-gram precisions of a generated sequence to the reference. Then, it measures the average modified n-gram precision. Since there are different implementations of BLEU scores, we select the most commonly used implementation proposed by Papineni et al. [39] in software document generation. This implementation is widely used in software documents generation evaluation [20, 23, 32].

4.1.2 METEOR. METEOR is proposed by Banerjee et al. [4] and is widely used to evaluate machine translation techniques. It evaluates translation hypotheses by aligning them to reference translations and calculating sentence-level similarity scores. Different from the BLEU score, it is a recall-oriented method that reflects how much the translated results cover the entire contents of the references. In this article, we use the most recently released version, namely, METEOR 1.5 [10], to evaluate the quality of the generated documents.

4.1.3 ROUGE-L. **Recall-Oriented Understudy for Gisting Evaluation** (**ROUGE**) is proposed by Lin et al. [27] and used to evaluate the quality of generated summaries. It evaluates the content adequacy by counting n-grams in the reference summaries that appear in generated summaries. It is also widely used in evaluating the quality of the generated software documents. Among different ROUGE scores, ROUGE-L is the most widely used metric in documentation generation tasks. It is calculated by using the longest common subsequence and F-measure.

[1] https://github.com/xing-hu/DocEvaluation.
[2] https://pypi.org/project/pycocoevalcap/.

Table 2. Criteria Details and Possible Answers used in the Human Evaluation Study

Criteria		Description	Possible Answers (on a scale between 1 and 5)
Language-related	Naturalness	Ignoring the content, considering the grammaticality and fluency of the documentation	1. Is very unfluent or has many grammatical errors that can hinder the reading of generated documentation 2. Is unfluent or has grammatical errors that can hinder the reading of generated documentation 3. Is somewhat fluent and has some grammatical errors that can not hinder the reading of generated documentation 4. Is fluent and has little grammatical errors that can not hinder the reading of generated documentation 5. Has no grammatical error and very fluent
	Expressiveness	Ignoring the content, considering the readability and understandability of the documentation's description	1. Is very hard to read and understand 2. Is hard to read and understand 3. Is somewhat readable and understandable 4. Is easy to read and understand 5. Is very easy to read and understand
Content-related	Adequacy	Considering the amount of contents carried from the source code/diff to documentation	1. Is missing a lot of very important information that can hinder the understanding of the source code/diff 2. Is missing some very important information that can hinder the understanding of the source code/diff 3. Is missing some information but the missing information is not necessary to understand the source code/*diff* 4. Is missing little information but the missing information is not necessary to understand the source code/diff 5. Is not missing any information
	Conciseness	Considering the amount of unnecessary information contained in documentation	1. Has a lot of unnecessary information 2. Has more unnecessary information 3. Has some unnecessary information 4. Has little unnecessary information 5. Has no unnecessary information
Effectiveness-related	Usefulness	Considering whether the documentation is useful for developers	1. Is totally useless 2. Is useless 3. Is somewhat useful 4. Is useful 5. Is very useful
	Understandability	Considering whether the documentation is helpful for developers to understand the source code/*diff*	1. Is totally unhelpful to understand the source code/*diff* 2. Is unhelpful to understand the source code/*diff* 3. Is somewhat helpful to understand the source code/*diff* 4. Is helpful to understand the source code/*diff* 5. Is very helpful to understand the source code/*diff*

4.1.4 CIDEr. **Consensus-based Image Description Evaluation (CIDEr)** is a consensus based evaluation protocol for image captioning [48]. It measures the similarity of a generated sentence against a set of ground truth sentences written by humans. It considers the frequency of n-grams in the reference sentences by computing the TF-IDF weighting for each n-gram. $CIDEr_n$ score for n-gram is computed using the average cosine similarity between the candidate sentence and the reference sentences. The final result is calculated by combining the scores for different n-grams (up to 4).

4.1.5 SPICE. **Semantic Propositional Image Caption Evaluation (SPICE)** [2] is a principled metric for automatic image caption evaluation that compares semantic propositional content. Different from other metrics that are sensitive to n-gram overlap, SPICE hypothesizes that semantic propositional content is an important component of human caption evaluation. SPICE measures how well caption generators recover objects, attributes, and the relations between them.

4.2 Human Evaluation Metrics

Generally, human evaluation metrics include three aspects, *Language-related*, *Content-related*, and *Effectiveness-related*. The scores of them are on a scale from 1 to 5 in which 1 means the worst and 5 means the best.

4.2.1 Language-related. Language-related metrics measure the generated documents considering the natural language features and ignoring the documentation's contents. It usually evaluates the generated sentences according to grammaticality and fluency. We select two metrics, namely, Naturalness and Expressiveness, to measure whether generated sentences are fluent and expressive.

— Naturalness [50]: measures the generated documentation considering fluency and grammaticality.
— Expressiveness [36, 37]: measures how readable and understandable the generated documentation.

4.2.2 Content-related. Content-related metrics measure generated documentation by considering the amount of contents carried from the input code to the generated documentation, ignoring the fluency of the text. We follow Sridhara et al. [44] and Mcburney et al. [35] to evaluate generated documentation by Content adequacy and Conciseness.

— Content adequacy [35, 44]: measures whether generated documentation misses some essential information that can hinder understanding the method.
— Conciseness [35, 44]: measures whether generated documentation contains unnecessary information.

4.2.3 Effectiveness-related. Effectiveness-related metrics are used to evaluate whether generated documentation is useful or helps developers understand programs.

— Usefulness [35]: is used to evaluate how useful the documentation is. For example, a comment can be marked as useful if it provides any useful information (e.g., implementation details and potential call risk) for developers.
— Code Understandability [43]: is used to evaluate to what extent the generated documentation can help developers understand programs.

5 MODELS

This section presents details of different models used in the two tasks. For each task, we select three state-of-the-art approaches and evaluate the documentation generated by them.

5.1 Comment Generation Models

5.1.1 Hybrid-DeepCom [20]. Hybrid-DeepCom is an approach that learns lexical and syntactical information at the same time while generating Java comments. It first converts the **Abstract Syntax Tree** (**AST**) into a sequence by a special sequential method, **Structure-based Traversal** (**SBT**). To address the vocabulary challenge, it splits identifiers into subtokens according to the camel naming convention. Then, it exploits two **Recurrent Neural Networks** (**RNNs**) to encode code tokens and AST sequences and uses the attention mechanism to fuse the lexical and structural information.

5.1.2 Code2Seq [1]. Code2Seq is proposed to encode source code by leveraging the syntactic structure of programming languages. The source code is represented as a set of compositional paths over its AST. Each path is encoded into a vector by the **Long Short Term Memory** (**LSTM**), and tokens (terminals' values) are encoded into sub-tokens embeddings. This work is applied to two tasks: code summarization (i.e., predict a Java method's name given its body) and code captioning (i.e., predict a natural language sentence that describes a C# snippet). In this article, we use it to generate code comments for Java methods.

5.1.3 Re^{2}Com [50]. Re^{2}Com is another state-of-the-art approach to generating code comments for Java methods. Different from previous studies, it integrates IR techniques and neural networks. Re^{2}Com mainly consists of two modules: a Retrieve module and a Refine module. It first exploits the Retrieve module to retrieve similar code snippets and their comments from the training set. The model then takes code snippets, similar code snippets, and retrieved comments (i.e., exemplars) as input and then generates comments. Re^{2}Com leverages the advantages of IR-based techniques and neural networks.

5.2 Commit Message Generation Models

5.2.1 NMT [23]. **Neural Machine Translation** (**NMT**) is neural networks that model the translation process from a source language sequence to a target language sequence. Jiang et al. [23] adopt the NMT algorithm to the commit message generation task. The model mainly consists of an Encoder and a Decoder. To deal with long diff sequences, they integrate the attention mechanism [3] into the NMT. They use the Theano-based framework Nematus to implement the NMT model. In this article, we reuse the results provided by Liu et al. [32].

5.2.2 NNGen [32]. NNGen leverages the **nearest neighbor** (**NN**) algorithm to produce commit messages. Compared to the NMT model, the NNGen is more straightforward and faster. It first selects top-k training diffs with the highest similarity scores by calculating the cosine similarity between the new diff vector and each training diff vector. Then, it selects the nearest neighbor with the highest BLEU-4 score from top-k training diffs. Finally, NNGen outputs the message of the nearest neighbor as the final result.

5.2.3 PtrGNCMsg [30]. Because the software developers are free to create any identifiers they like, the **Out-of-Vocabulary** (**OoV**) problem in modeling source code is more serious than that in the NLP tasks. Liu et al. [30] propose a PtrGNCMsg approach that is based on the pointer-generator network [41] to deal with the OoV issue while translating diffs into commit messages. It is an adapted version of the attentional RNN Encoder-Decoder model. While generating commit messages, PtrGNCMsg can either copy words from diffs or generate words from the vocabulary. In this way, it can generate more accurate commit messages with OoV identifiers by copying them from diffs.

6 RESULTS

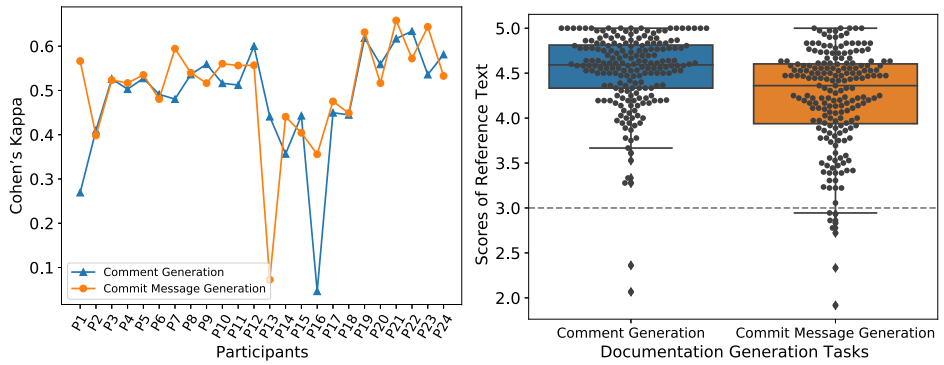
6.1 RQ1: Comparison of Results of the Automatic Evaluation and Human Evaluation

We first compare the results from the automatic evaluation and human evaluation. Table 3 and Table 5 summarize the overall performance of different approaches in terms of automatic and human scores.

6.1.1 Results on Automatic Metrics. The experimental results of the two documentation generation tasks are shown in Table 3. We follow previous studies [1, 20, 23, 30, 32, 50] to calculate the gap between the documentation generated by different methods and the ground truth. The gap is measured by BLEU, METEOR, ROUGE-L, CIDEr, and SPICE. According to articles [2, 10, 27, 39, 48] that propose these metrics, BLEU and METEOR return scores at the corpus level, whereas ROUGE-L, CIDEr, and SPICE return average scores from each case. Thus, we give the standard deviation (shown in the brackets) for ROUGE-L, CIDEr, and SPICE. We can observe that these metrics are different in reflecting the ability of different models. For the code comment generation task, Re^{2}Com outperforms the other two approaches considering BLEU, METEOR, and ROUGE-L. However, Hybrid-DeepCom performs best in terms of CIDEr and Code2Seq performs best in terms of SPICE. For the commit message generation task, NNGen performs the best among all approaches.

Table 3. Comparison Results of Different Approaches on Automatic Metrics

Approaches	BLEU(%)	METEOR(%)	ROUGE-L(%)	CIDEr	SPICE(%)
Comment Generation Task					
Hybrid-DeepCom	21.01	15.87	33.10 (0.31)	**1.85** (3.40)	21.41 (0.33)
Code2Seq	15.56	14.43	30.77 (0.26)	1.34 (2.57)	**22.13** (0.28)
Re^{2}Com	**21.06**	**16.04**	**36.20** (0.30)	1.70 (3.34)	21.75 (0.35)
Commit Message Generation Task					
NMT	14.19	12.99	23.66 (0.29)	1.06 (2.34)	18.07 (0.33))
NNGen	**16.42**	**14.03**	**27.17** (0.33)	**1.37** (2.74)	**20.58** (0.34)
PtrGNCMsg	12.31	11.94	24.45 (0.28)	1.10 (2.29)	17.38 (0.30)



(a) The agreement of different participants according to Cohen's Kappa.

(b) Human evaluation score distribution of reference text

Fig. 5. The bias analysis of human evaluation.

6.1.2 Results on Human Evaluation Metrics. In this article, we ask participants to score each documentation on a 5-point Likert scale. We analyze the results from human annotators.

(1) Bias detection. Human evaluation is typically treated as the gold standard for assessing the performance of automatic evaluation metrics. However, quality assessment is known to be a complex tasks with low levels of agreement between annotators [14]. Thus, we analyze the bias and consistency among different annotators.

We first analyze the agreement levels among annotators by using Cohen's Kappa (κ). Figure 5(a) illustrates the average Cohen's Kappa scores of each participant with others. According to Fleiss et al. [12], kappas over 0.75 are excellent, 0.40–0.75 are fair to good, and below 0.40 are a poor. For the comment generation task, annotators P1 and P16 have poor agreement ($\kappa < 0.4$) with others. For the commit message generation task, annotators P2, P13, and P16 have a poor agreement ($\kappa < 0.4$) with others. We follow Liu et al. [29], and we exclude their responses from the analysis below.

Then, we analyze the bias introduced by reference text. While comparing the generated documentation and reference text, the quality of reference text should be guaranteed to have good quality. In this article, comments and commit messages written by developers are taken as a reference text. To ensure the correlation is more reliable, we exclude reference text that achieves low scores from annotators from the analysis below. We compute the average human evaluation scores (after normalizing personal bias) of the reference text that is shown in Figure 5(b). Two reference comments achieve low scores from annotators. For reference commit messages, 18 references have low scores. These low-quality reference texts may introduce bias while computing the correlation.

Table 4. The Average Kendall Correlation (τ) and Pearson Correlation (r) Among Human Evaluators

Metrics	*Naturalness*		*Expressiveness*		*Adequacy*		*Conciseness*		*Usefulness*		*Understandability*	
	τ	r	τ	r	τ	r	τ	r	τ	r	τ	r
Comment Generation	0.47	0.51	0.44	0.49	0.72	0.81	0.66	0.74	0.71	0.79	0.71	0.78
Commit Message Generation	0.58	0.74	0.57	0.71	0.66	0.76	0.66	0.75	0.66	0.74	0.65	0.74

Table 5. Comparison Results of Different Approaches on Human Evaluation

Approaches	*Language-related*		*Content-related*		*Effectiveness-related*	
	Naturalness	**Expressiveness**	**Adequacy**	**Conciseness**	**Usefulness**	**Understandability**
Comment Generation Task						
Hybrid-DeepCom	4.60 (4.65)	4.21 (4.28)	2.57 (2.54)	2.51 (2.47)	2.26 (2.22)	2.12 (2.12)
Code2Seq	4.42 (4.46)	4.01 (4.04)	1.48 (1.41)	1.53 (1.48)	1.32 (1.24)	1.28 (1.19)
Re^2Com	4.52 (4.57)	3.41 (4.26)	3.21 (3.26)	3.07 (3.11)	2.80 (2.90)	2.71 (2.78)
Reference	4.73 (4.80)	4.19 (4.63)	4.45 (4.56)	4.31 (4.41)	4.28 (4.40)	4.22 (4.33)
Commit Message Generation Task						
NMT	4.41 (4.49)	4.21 (4.3)	2.44 (2.46)	2.45 (2.5)	2.14 (2.15)	2.11 (2.12)
NNGen	4.46 (4.51)	4.27 (4.32)	2.57 (2.61)	2.56 (2.60)	2.28 (2.30)	2.23 (2.26)
PtrGNCMsg	3.53 (3.61)	3.41 (3.48)	2.66 (2.71)	2.68 (2.71)	2.42 (2.46)	2.39 (2.42)
Reference	4.68 (4.77)	4.61 (4.72)	4.02 (4.25)	4.00 (4.21)	3.85 (4.05)	3.76 (3.98)

Reference means the human-written documentation. Scores in brackets mean the results removing bias.

For example, an uninformative reference text could result in a high BLEU score if an approach does generate similar, uninformative, documentation. This, in turn, would result in a low score for the generated documentation given by the human evaluators. The low score from evaluators together with the high BLEU score, will lower the correlation. Thus, we exclude cases whose reference text has low quality.

(2) Annotators Correlation. To compare the human evaluation consistency, we compute the Kendall Correlation and Pearson correlation among annotators after removing bias. Table 4 shows the correlation among annotators. For the comment generation task, annotators have low correlation (in terms of Kendall and Pearson correlations) on *Naturalness* and *Expressiveness*. Annotators have high Kendall and Pearson correlations in evaluating the *Adequacy*, *Conciseness*, *Usefulness*, *and Understandability* of generated comments except for the Kendall correlation on the *Conciseness*. For commit message generation task, annotators have moderate Kendall correlations ($0.5 \leq \tau < 0.7$) but high Pearson correlation ($0.7 \leq |r| < 0.9$) on evaluating all human evaluation metrics.

(3) Human Evaluation Results. The human evaluation results of these approaches are shown in Table 5. The scores in brackets are results without bias analyzed above and scores outside brackets are raw results with bias. After removing bias, the *Language-Related* results of all code comment generation approaches improve compared to results with bias. For *Content-Related* and *Effectiveness-Related* metrics, the results of Re^2Com and Reference improve whereas Hybrid-DeepCom and Code2Seq decrease after removing bias. In addition, all results of the commit message generation task improve after removing bias.

For the comment generation task shown in Table 5, Hybrid-DeepCom performs the best while considering the *Language-related* metrics. It means the text generated by Hybrid-DeepCom is much more fluent and has fewer grammar errors. The three models' Naturalness is more than 4, indicating that all models can generate fluent and grammatical comments. Considering *Content-related* and *Effectiveness-related* metrics, Re^2Com achieves the best performance and significantly outperforms Hybrid-DeepCom and Code2Seq. The text generated by the Re^2Com model expresses more content information (e.g., keywords and code structure) of Java methods. Besides, comments generated by Re^2Com are more useful and helpful in code understanding than those generated by others.

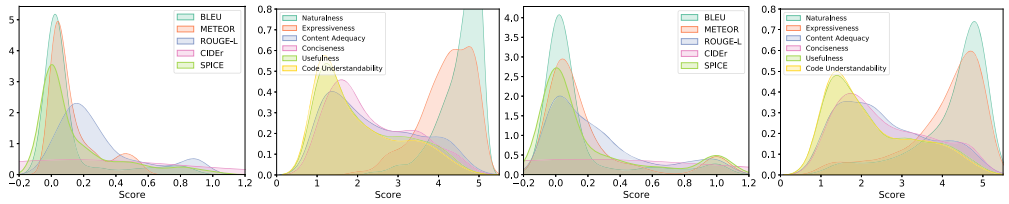


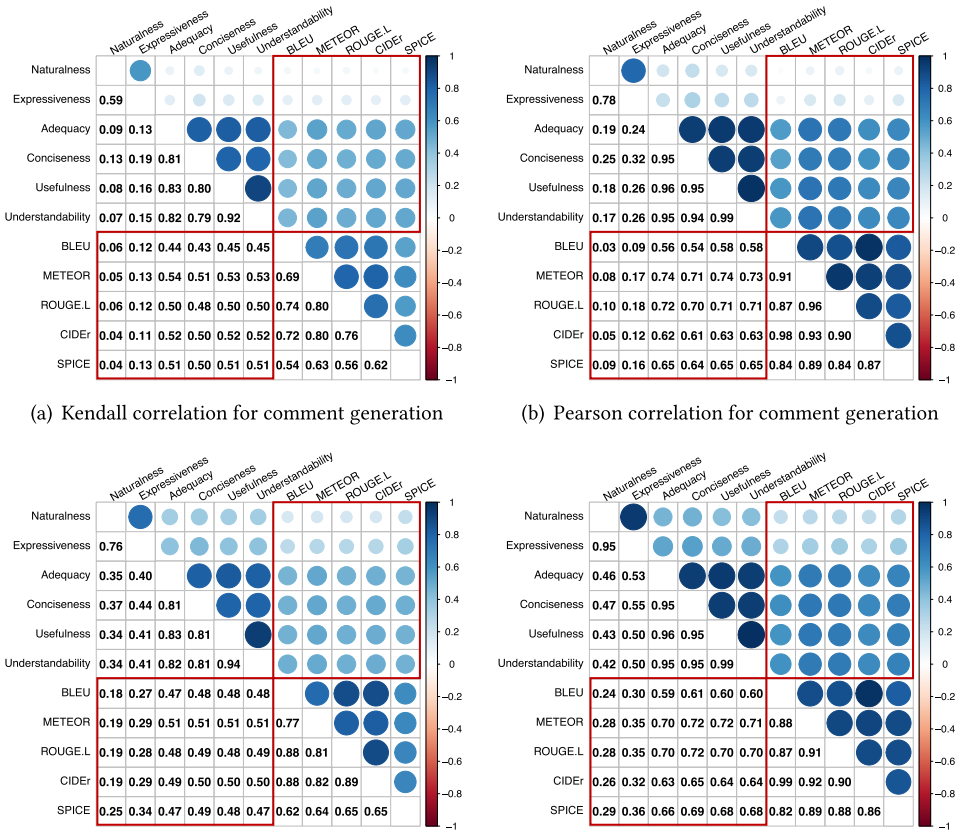
Fig. 6. Distribution of different metrics. Figure 6(a) and (b) are distributions of automatic metrics and human evaluation metrics for comment generation task, respectively; Figure 6(c) and (d) are distributions of automatic metrics and human evaluation metrics for commit message generation task.

For the commit message generation tasks, NNGen performs the best on *Language-related* metrics, whereas PtrGNCMsg performs the best on *Content-related* and *Effectiveness-related* metrics. The performance of PtrGNCMsg is much worse than the NMT model and the NNGen model considering *Language-related* metrics. But in terms of *Content-related* and *Effectiveness-related* metrics, it slightly outperforms other models. The messages generated by PtrGNCMsg contain more OoV words, although this article exploits the pointer mechanism to alleviate this issue. OoV words have negative impacts on naturalness and expressiveness. However, the pointer mechanism helps with the content information selection that contributes to code understandability.

From Table 5, we can find that documentation generated by different approaches achieve almost equivalent results to human-written documentation (i.e., Reference) in *Language-Related* aspects. In other words, generated documentation is easy to read and understand, and human annotators are hard to distinguish generated documentation from reference text considering the naturalness and expressiveness. This is a "milestone" for automated approaches as they can generate fluent, grammatical, and expressive documentation which is important in text generation tasks [46]. The next milestone is the achievements on the *Content-Related* and *Effectiveness-Related* aspects. To achieve this "milestone", the generated documentation should be semantic correct and consistent with the source code or commits.

6.1.3 Distribution of Different Metric Scores. We represent the distribution of different metric scores using the **Kernel Density Estimate** (**KDE**) plot. Shown as in Figure 6(a), BLEU and METEOR scores have similar distribution except for the second peak (BLEU is around 0.8 and METEOR is around 0.5). Compared to BLEU, METEOR, and SPICE the distribution of ROUGE-L has less variance, and its peaks are around 0.2 and 0.9. The commit message generation task is shown in Figure 6(c); BLEU, METEOR, ROUGE-L, and SPICE share similar peak positions at around 0.1 and 1.0, respectively. We can observe that automatic metrics tend to give a relatively low or a relatively high score, causing two peaks in Figure 6(a) and (c). Different from automatic metrics, the distribution as shown in Figure 6(b) and (d) of human evaluation metrics is quite different. *Content-related* and *Effectiveness-related* metrics have a similar distribution. Peaks of *Content-related* metrics and *Effectiveness-related* metrics are around 1.5 and 2, respectively, for both tasks. However, the peaks of *Language-related* metrics are much higher than other metrics.

Summary of RQ1: We replicate the state-of-the-art approaches for documentation generation and evaluate them in terms of automatic and human evaluation metrics. We also analyze the bias in the annotated data and the correlation between annotators can reach 0.81 in Pearson Correlation and 0.72 in Kendall Correlation.



(a) Kendall correlation for comment generation

(b) Pearson correlation for comment generation

(c) Kendall correlation for commit message generation

(d) Pearson correlation for commit message generation

Fig. 7. Kendall and Pearson correlation results for comment generation task and commit message generation task. Bordered area shows correlations between human ratings and automatic metrics, the rest shows correlations among the metrics. Blue colour of circles indicates positive correlation, while red indicates negative correlation. The size of circles denotes the correlation strength.

6.2 RQ2: Correlation inside of Human Evaluation or Automatic Evaluation Metrics

In this RQ, we explore the metric correlation inside of human evaluation or automatic evaluation. The part outside the box of Figure 7 shows the Kendall and Pearson correlation results among evaluation metrics for comment generation task and commit message generation task.

6.2.1 Human Evaluation Metrics Consistency. For the comment generation task, *Naturalness* and *Expressiveness* have moderate Kendall correlation and high Pearson correlation between them. In addition, *Language-related* metrics (i.e., *Naturalness* and *Expressiveness*) negligibly correlate to other metrics ($\tau|r < 0.3$). Considering the high scores of the naturalness and expressiveness in Table 5, we find that most comment generation models can generate fluent and grammatical comments. In other words, we can not measure the quality of different models only by considering whether the generated comments are fluent and grammatical or not. For *Content-Related* metrics (i.e., *Adequacy* and *Conciseness*) and *Effectiveness-Related* metrics (i.e., *Usefulness* and *Understandability*), they have a high or very high correlation between each other.

For the commit message generation task, *Naturalness* and *Expressiveness* have high Kendall correlation (τ : 0.76) and a very high Pearson correlation (r : 0.95) between them. *Naturalness* and

Expressiveness have low Kendall correlations (with $0.34 \leq \tau < 0.44$) with other metrics. In addition, *Naturalness* has low Pearson correlation and *Expressiveness* has a moderate Pearson correlation with other metrics. The correlation between *Content-related* metrics and others are similar to the comment generation task. We can find that generated documentation with exhaustive content and less useless information is much more useful for developers. In other words, the more information is carried from the input source code/commit diff, the more useful and helpful the generated documentation is.

6.2.2 Automatic Metrics Consistency. Among these five automatic metrics (i.e., BLEU, METEOR, ROUGE-L, CIDEr, and SPICE), SPICE has the lowest correlation with others. SPICE has moderate Kendall correlation and high Pearson correlation with other automatic metrics on the two tasks. Other automatic metrics have high correlation (with $\tau | r > 0.7$) from each other. SPICE evaluates the similarity of candidate and reference text by comparing the semantic relations in scene graphs instead of N-gram overlappings. It causes the lower correlation between SPICE and other automatic metrics. Automatic metrics (except SPICE) are generally based on n-gram overlaps between the two sentences, thus, they achieve a high correlation. Particularly, CIDEr almost perfect Pearson correlates to BLEU (with $r : 0.98$ and $r : 0.99$) for comment generation task and commit message generation task, respectively.

Summary of RQ2: For human evaluation metrics, we find that *Content-Related* metrics (i.e., Adequacy and Conciseness) and *Effectiveness-Related* metrics (i.e., Usefulness and Understandability), the have high or very high correlation between each other. It means that the more information is carried from the input source code/commit diff, the more useful and helpful the generated documentation is. For automatic metrics, they have high correlation from each other.

6.3 RQ3: Do Automatic Metrics Such as BLEU, METEOR, ROUGE-L, CIDEr, and SPICE Correlate with Human Judgment on Generated Documentation?

In RQ2, we mainly explore correlations in human evaluation metrics and automatic metrics, respectively. In this RQ, we further analyze the correlation between automatic metrics and human evaluation metrics. This correlation reveals how much automatic metrics can reflect the human perspective.

6.3.1 Correlation Between Automatic Metrics and Human Evaluation Metrics. Figure 8 illustrates the relationship between automatic metrics and human evaluation metrics in comment generation tasks via a Scatter plot. The data points of *Language-Related* scores are scattered close to 4–5 and we can not find an obvious relationship between it and automatic metrics. When it comes to *Content-Related* and *Effectiveness-Related* metrics, these points are scattered more regularly. We can observe a fairly even linear band of data points in the relationship between METEOR/ROUGE-L and *Content-Related*/*Effectiveness-Related* metrics. The larger bandwidths of ROUGE-L and *Content-Related*/*Effectiveness-Related* metrics indicate that ROUGE-L is less correlated to them. The scatter plot of commit message generation task shown in Figure 9 is similar to Figure 8.

To further explore the detailed correlation between automatic metrics and human evaluation metrics, we follow Coughlin et al. [9] and measure Kendall Correlation τ and Pearson Correlation r [5]. The results are illustrated in the bordered area of Figure 7. Automatic metrics are weakly correlated to *Language-related* metrics (both Naturalness and Expressiveness). Specifically, they have no correlation with the Naturalness for comment generation task (Pearson's $r < 0.1$ and Kendall $\tau < 0.1$, $p < 10^{-3}$). For *Language-related* metrics and *Effectiveness-related* metrics, the Kendall

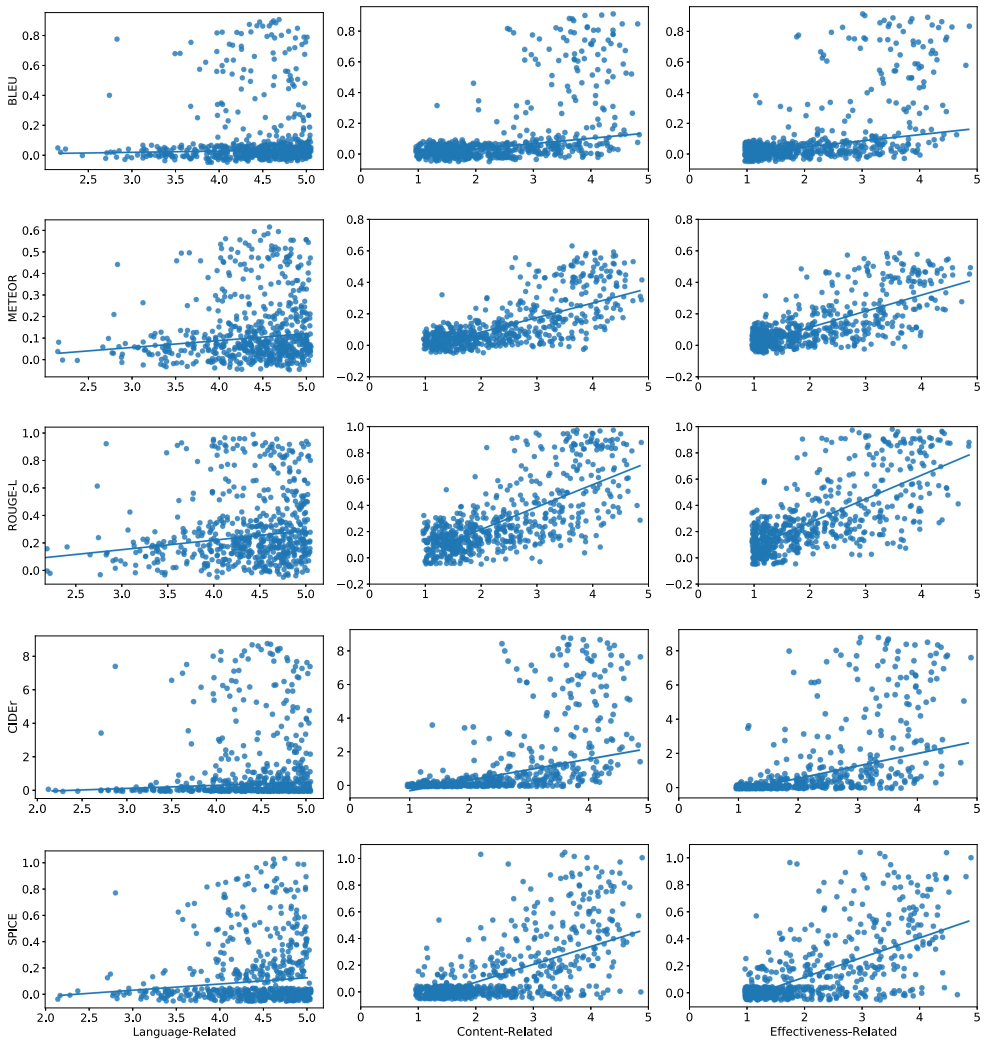


Fig. 8. Scatter plots of automatic metrics scores against the human evaluation metrics scores on code comment generation task.

correlations between automatic metrics and them (varies from 0.43 to 0.54) are much lower than those of human evaluators (varies from 0.66 to 0.71). Although the Pearson correlations between automatic metrics and *Language-related* metrics and *Effectiveness-related* metrics are moderate, they are also much lower than those of human evaluators (varies from 0.74 to 0.81). In addition, we find that the METEOR achieves the strongest correlation to human evaluation metrics, whereas BLEU has the weakest correlation with them. However, METEOR is often omitted by many code documentation works [1, 32, 50]. According to Figure 7 shows above, METEOR (with Pearson's r about 0.7 and Kendall's τ about 0.5, $p < 10^{-3}$) should be included among the current widely used metrics to measure the documentation generation task.

6.3.2 Correlations in Different Quality Levels. As suggested by Novikova et al. [38], we "bin" our annotated data into three groups: bad, which comprises low ratings (≤2); good, comprising

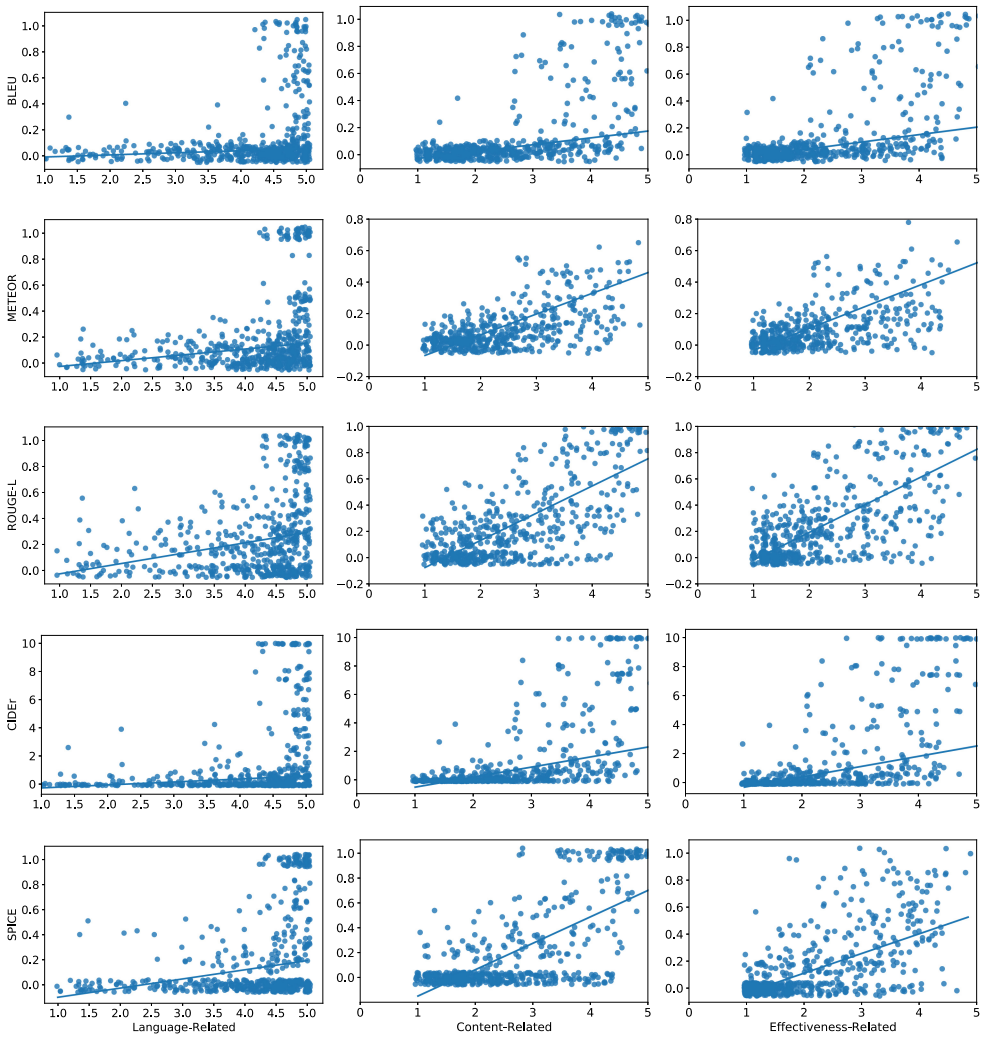


Fig. 9. Scatter plots of automatic metrics scores against the human evaluation metrics scores on commit message generation task.

high ratings (≥4); and finally a group comprising neutral ratings. Then, we conduct a correlation analysis between automatic metrics and human metrics in these three groups, respectively.

The top part of Table 6 shows the correlations of automatic metrics and human ratings of code comment generation task in different bins. We find that there are no generated comments with low human ratings of *Naturalness*. The generated comments with high human rating scores and low human rating scores of *Effectiveness-Related* metrics correlate significantly better with automatic metrics than those with average human ratings. It shows that metrics are good in distinguishing extreme cases, i.e., cases rated as clearly good or bad by the human judges, but do not perform well for cases rated in the middle of the Likert scale. However, the *Content-Related* correlations in the average bin are better than in the bad bin. The highest correlation in the bad bin barely reaches $r \leq 0.26$ (a very weak correlation). For the commit message generation task, the same pattern can be observed for correlations.

Table 6. Pearson Correlation Between Metrics and Human Ratings for Results from Different Bins

		Bad						Avg						Good					
Comment Generation Task		*Nat*	*Exp*	*Ade*	*Con*	*Use*	*Und*	*Nat*	*Exp*	*Ade*	*Con*	*Use*	*Und*	*Nat*	*Exp*	*Ade*	*Con*	*Use*	*Und*
	BLEU	-	0.05	0.13	0.12	0.16	0.18	0.20	−0.03	0.26	0.20	0.05	0.19	0.02	0.04	0.32	0.23	0.32	0.30
	METEOR	-	−0.95	0.26	0.24	0.35	0.37	0.22	−0.03	0.37	0.30	0.07	0.18	0.06	0.11	0.43	0.31	0.35	0.32
	ROUGE-L	-	−0.39	0.24	0.23	0.33	0.36	0.32	−0.04	0.33	0.24	0.01	0.13	0.07	0.11	0.38	0.28	0.29	0.25
	CIDEr	-	−1.0	0.15	0.11	0.21	0.22	0.17	−0.07	0.30	0.19	0.07	0.17	0.04	0.07	0.36	0.29	0.30	0.28
	SPICE	-	-	0.13	0.13	0.27	0.30	0.08	0.06	0.28	0.26	0.13	0.19	0.07	0.10	0.35	0.30	0.30	0.28
		Bad						**Avg**						**Good**					
Commit Message Generation Task		*Nat*	*Exp*	*Ade*	*Con*	*Use*	*Und*	*Nat*	*Exp*	*Ade*	*Con*	*Use*	*Und*	*Nat*	*Exp*	*Ade*	*Con*	*Use*	*Und*
	BLEU	0.01	0.03	0.04	0.06	0.20	0.23	−0.19	−0.24	0.11	0.20	0.04	−0.02	0.24	0.34	0.47	0.42	0.35	0.39
	METEOR	0.13	0.22	0.20	0.18	0.32	0.35	−0.02	0.07	0.15	0.27	0.13	0.05	0.26	0.38	0.60	0.60	0.49	0.48
	ROUGE-L	0.11	0.04	0.13	0.11	0.28	0.32	−0.02	0.00	0.16	0.18	0.13	0.03	0.25	0.40	0.52	0.49	0.45	0.39
	CIDEr	−0.03	0.08	0.10	0.14	0.22	0.26	−0.22	−0.19	0.10	0.23	0.09	−0.02	0.25	0.35	0.52	0.48	0.40	0.44
	SPICE	0.04	0.05	0.06	0.13	0.13	0.17	0.05	0.12	0.14	0.26	0.13	0.02	0.29	0.41	0.50	0.52	0.47	0.40

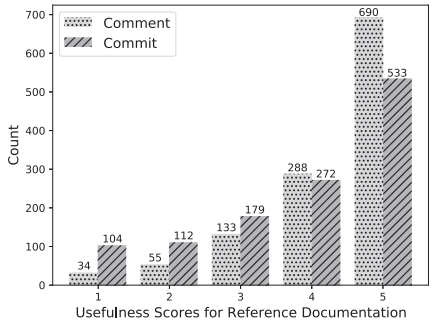


Fig. 10. The statistic information of *Usefulness* scores for Reference documentation (i.e., human-written documentation).

Summary of RQ3: Among all automatic metrics, METEOR has the highest correlation with human evaluation metrics (Pearson: 0.7 and Kendall: 0.5). Although this correlation is good, it still has a great gap when compared to correlation between different annotators (Pearson: 0.81 and Kendall: 0.72).

7 DISCUSSION

7.1 How do Developers Evaluate the Usefulness of the Code Documentation?

Although reference documentation achieves high usefulness scores (shown in Table 5), we find that some cases have low-scored reference text. According to Novikova et al. [38], ratings less than 2 (≤ 2) are regarded as low score ratings. As Figure 10 shows, 89 and 216 cases in the 1,200 labeled reference documentation (200 reference documentation labeled by six annotators) with the usefulness score lower than 3 (i.e., 1–2) for comment generation task and commit message generation task, respectively. It indicates that about 7.4% to 18% of human-written reference texts are not deemed to be highly useful by other developers. To investigate the developers' perspectives on the usefulness of code documentation, we conduct an interview with 24 annotators again and ask them how to evaluate the usefulness and why they give low scores for the documentation usefulness.

According to annotators' responses, we summarize three reasons that they score low usefulness of code documentation:

Java method:

```
private static Map<INaviViewNode,INaviViewNode> createNodes(final INaviView target,
                                                    final Collection<INaviViewNode>nodes){
    final HashMap<INaviViewNode,INaviViewNode>map=
                                          new HashMap<INaviViewNode,INaviViewNode>();
    for(final INaviViewNode blockNode:nodes){
        createNodes(target,blockNode,map);
    }
    return map;
}
```

Reference: clones a list of source nodes and inserts them into the target view.
One of the usefulness score: 3

Fig. 11. A case with reference text receiving a low usefulness score in the survey.

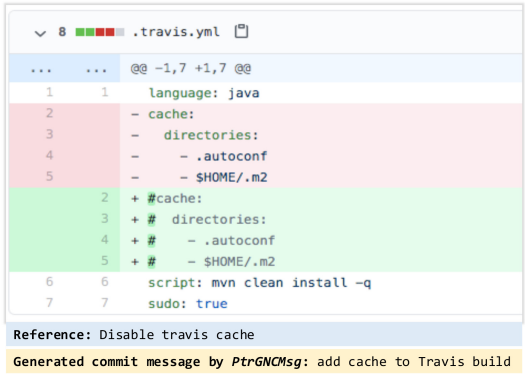


Fig. 12. An example of generated commit message in which key information is wrong.

Reason 1. Code Understandability. The usefulness of code comment is depending on the understandability of the source code. If the source code is hard to understand, the comment is useful to help developers understand the source code. On the contrary, the comment is useless and unnecessary if the source code is very easy.

Reason 2. Program Comprehension Improvement. According to annotators, a useful comment should help improve code comprehensibility. In particular, it can tell developers the functionality and how to use a code snippet. If the comment misses such information, it would be useless for developers.

Reason 3. Lack of Keywords. For some annotators, keywords are one of the most important factors for them to rate the Usefulness. Comments are useful if all important keywords in the source code are listed in them. In other words, the lack of keywords to illustrate what the programs do in the code comments leads to a low scores (just like Figure 11).

In addition, the developers' perspective on the commit message is different. According to their responses, the reasons they give low scores for commit message usefulness are as follows:

Reason 1. Trivial Commit Messages. A useful commit message should include code change details that explain what changed. Trivial commit messages such as "Create README.md" and "Fix typo" are limited for developers to know the main content of commits. Therefore, trivial commit messages obtain low usefulness scores.

Reason 2. Wrong Keywords. For annotators, keywords are essential for developers to read commit messages. Only one wrong keyword may cause low usefulness scores. For example, Figure 12 shows one case that contains the wrong keyword. The details included in the generated message "add cache to Travis build" is wrong; the correct one is "disable travis cache". The wrong detail leads to a low usefulness score.

According to the annotators' responses, key pieces of information are essential for the code documentation's usefulness. Thus, the current automatic metrics should be improved by considering the number of keywords that explain the source code or *diff*.

7.2 Code Documentation Generation vs. Neural Machine Translation (NMT)

Deep-learning-based documentation generation techniques are typically regarded as a kind of translation task, translating from one (programming) language to another (natural) language [15]. Papineni et al. [39] reported that for the task of translating from Chinese to English, the BLEU score and the human judgment is about 0.96 (the bilingual group). The correlation is much higher than what our study finds for code documentation generation task.

We analyze the code documentation generation and NMT task (e.g., translating English to French), and find that:

— Developers have different perspectives on code documentation. For example, they may judge a commit message by considering "what" (what content is modified), "How" (how to modify), "Why" (why to modify). The human evaluation judgment has significant subjective differences. However, current automatic metrics can not reflect these subjective differences.
— The corpus of code documentation generation is not aligned (i.e., each word in generated sentence has a corresponding word in the source sentence [3]). The generated documentation is abstracted from the input software artifacts. In addition, the generated documentation can describe the functionality of the source code or how to use the source code.
— There is only one reference when evaluating the generated documentation, whereas there are multiple references in the NMT task. In the documentation, different words may describe the same contents. Therefore, multiple references can be included while evaluating generated documentation.

7.3 Gap Between Machine-generated Documentation and Human-written Documentation

In addition, we find that the results of generated documentation on *Content-related* and *Effectiveness-related* metrics are significantly worse than the results for the reference text (with $p\text{-}value < 10^{-3}$). We further analyze the distribution of scores in the two aspects. The distribution is shown in Figures 13 and 14. The scores of most human-written references vary from 3.5 to 4.5 indicate that references are more helpful for developers to understand the source code/*diffs*. As expected, the generated documentation shows fairly low scores (most scores are less than 3). In particular, most scores of comments generated by Code2Seq are lower than 2. It demonstrates that many generated comments miss the most important information to describe the source code.

The results demonstrate a big gap between the machine-generated documentation and human-written documentation considering the *Content-related* and *Effectiveness-related* aspects.

7.4 Usefulness vs. Understandability

According to the high correlation between Usefulness and Understandability shown in Figure 7 and the discussion of the Usefulness criteria in Section 7.1, many developers think Usefulness and Understandability are equivalent. However, from the interview of annotators, we find that they are not always equivalent.

— *Implementation details are not always useful.* Implementation details describe the algorithms of code snippets and can improve the understandability of source code. However, some developers think them useless as they want more information on the "how to use" information. If the documentation is just a "translation" of the source code, it is not useful

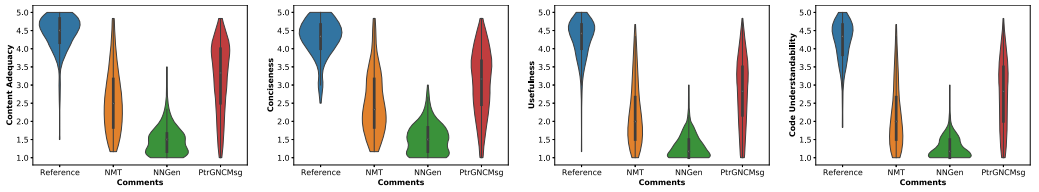


Fig. 13. Distribution of the Content related and Effectiveness related results of comments generated by different approaches.

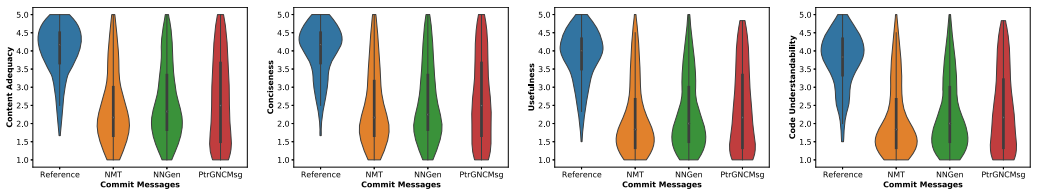


Fig. 14. Distribution of the Content related and Effectiveness related results of commit messages generated by different approaches.

— *Source code is self-documenting*. Sometimes, developers have no problem reading code especially for senior developers, and in fact, preferred it for finding more accurate information. Thus, documentation (even understandable) is useless for them.

7.5 Implications

According to Figure 7, we can observe that *Usefulness* and *Understandability* are most relevant to *Adequacy*. In other words, the *Usefulness* and *Understandability* of the generated documentation depend on whether it contains enough useful information carried from the input. Therefore, we can improve existing automatic metrics by considering the input source code. First, we should identify tokens that are related to documentation in the input, such as identifiers and APIs in the source code. Next, we compute the overlapping rate between generated documentation and the identified tokens. Finally, we get the final score by integrating existing automatic metrics and the overlapping rate. In this way, the improved automatic scores can illustrate the *Adequacy* metric, thus improving the correlations.

8 THREATS TO VALIDITY

One threat to validity is that both the two code documentation tasks are based on Java datasets. Although Java may not be representative of all programming languages, models we used in this article can be easily applied to other programming languages.

The second threat to validity is our human evaluation. We cannot guarantee that each score assigned to each case is fair. Besides, participants' programming experiences may introduce bias into the evaluation. To mitigate this threat, we evaluate each case by six human annotators, and we use the average score of the six annotators as the final score for each case.

The third threat to validity is the amount of cases that practitioners evaluate. In this article, each annotator evaluates 50 cases for each of the two tasks. Thus, each annotator evaluates 100 cases and each case contains four items. We follow Liu et al. [29] which give each annotator 100

questions per dataset and each question contains 5 different responses. A Long-time labeling task may lead to exhaustion of the annotators and then result in a decrease in the labeled data quality. To alleviate this threat, we recommend that participants rest for at least half an hour for every half hour annotation. In addition, we do not limit the amount of time for annotators to complete this user study. Most practitioners complete the evaluation process in one week.

Another threat to validity is the replication of each model. To ensure that the experimental results are consistent with their articles, we reuse the generated documentation if it is provided [32]. If we cannot get the generated documentation, we retrain the models using the source code provided by the authors. In addition, we reuse the parameters provided by the authors and do not conduct parameter tuning. The performance of deep learning models can be affected by tuning parameters. Thus, the quality of generated documentation by deep learning models may vary along with parameters. We compare the results generated by models with parameters provided by original articles. According to previous studies [1, 50], parameters used in their works are tuned and best parameters are selected according to the performance. Therefore, reusing the provided parameters can ensure the quality of the generated documentation.

9 RELATED WORK

9.1 Code Comment Generation

Code comment is one of the most common code documentation and can help developers understand what a program does. However, few software projects adequately comment on the code [44]. Therefore, recent research has made great efforts on the automatic generation of comments from source code.

Traditional techniques mainly generate code comments by defining heuristic rules and templates. Sridhara et al. [44] propose **Software Word Usage Model** (**SWUM**) to capture key terms in source code and then define different templates for different semantic segments in source code to generate readable comments. Similarly, Mcburney et al. [35] use SWUM to capture keywords in the source code and utilize PageRank to select important Moreno et al. [36] pre-define heuristic rules to select information and generate comments for Java classes by combining the information. Haiduc et al. [17] exploit IR techniques, Vector Space Model, and Latent Semantic Indexing to select relevant terms from source code and assemble them into comments. These studies usually evaluate their approaches through human evaluation, for example, Moreno et al. [36] ask 22 graduate students to judge the content adequacy, conciseness, and expressiveness of automatically generated summaries for 40 Java classes.

Recently, deep learning techniques are widely used to generate comments from source code. Generally, they exploit neural networks with Encoder-Decoder architecture in which the Encoder learns the representation of source code and the Decoder generates natural language descriptions from the representation. Compare to the traditional techniques, these techniques do not need manually selecting keywords and defining heuristic rules. Iyer et al. [22] propose to generate comments for C# and SQL code snippets by an RNN equipped with an attention mechanism. Hu et al. [19, 20] integrate the AST sequences into the Seq2Seq model to generate code comments for Java methods and achieve better results. Leclair et al. [25] then propose to use multi-encoders to deal with the AST sequences and the source code. Alon et al. [1] propose a novel technique Code2Seq that encodes AST paths and generates more accurate comments. Recently, researchers have taken advantage of both the retrieval technique and the neural network [50, 53]. Wei et al. [50] propose Re^{2}Com that generates comments with the assistance of similar code snippets and their comments. Zhang et al. [53] propose *Rencos* to combine the retrieval-based and NMT-based methods. Liu et al. [31] propose a retrieval-augmented mechanism to generate comments for C projects. They propose a

framework named HGNN to fuse the static graph and dynamic graph to capture graph information. LeClair et al. [24] propose a graph-based neural architecture that better matches the default structure of the AST to generate comments. In addition, reinforcement learning is also exploited to improve the comment generation [49] by solving the exposure bias problem during generating comments.

Although human evaluation is accurate and convincing, it is time-consuming and high costs. Therefore, these studies usually use automatic metrics (e.g., BLEU and ROUGE) to evaluate on a large-scale dataset.

9.2 Commit Message Generation

Rule-based approaches use pre-defined rules and templates to generate commit messages. DeltaDoc [6] analyze the program's control flow between different code versions and then use the template "do Y Instead of Z" to generate the commit messages. ChangeScribe [8, 28] also fill a predefined commit message template with extracted information from corresponding source code changes and the abstract syntax trees.

Several retrieval base approaches were proposed for the effectiveness of information retrieval techniques. Huang et al. [21] reuse the commit messages by measuring the syntactic similarity and semantic similarity between changed code fragments. Different from Huang et al. [21], Liu et al. [32] not only focus on code changes and generate commit messages directly from git diffs based on the nearest neighbor algorithm.

Deep learning models have been used to generate commit messages for git diffs in recent years. Jiang et al. [23] and Loyola et al. [33, 34] generate commit messages based on the attentional encoder-decoder model, which is a classic method in Natural Language Processing. PtrGNCMsg [30] uses a pointer-generator network to address the OoV problem. In addition to solving the OoV problem, CODISUM [52] combines the structure and semantics of code to generate commit messages.

10 CONCLUSION

Developers spend a great deal of time on program comprehension during software maintenance and development. To help developers better understand programs, researchers have proposed various methods to automatically generate documentation. Specifically, many deep learning approaches have been proposed to learn the representation of source code and then generate code documentation. They usually evaluate models' performance by automatic metrics that are widely used in the NLP domain. However, there is no study to explore whether these metrics are correlated to human evaluation metrics.

In this article, we replicate and evaluate the state-of-the-art documentation generation approaches. Then, we analyze the correlation between the automatic results and human evaluation results (based on ratings given by 24 user study participants). We find that documentation with higher content adequacy and conciseness scores is much more useful and helpful for code understandability. Also, our study finds that among the automated metrics, METEOR is the most correlated to human evaluation metrics. In future work, we will improve automatic metrics and make them more correlated to human perspectives on code documentation generation.

REFERENCES

[1] Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. 2018. Code2seq: Generating sequences from structured representations of code. In *Proceedings of the International Conference on Learning Representations*.

[2] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. 2016. Spice: Semantic propositional image caption evaluation. In *Proceedings of the European Conference on Computer Vision*. Springer, 382–398.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR'15), San Diego, CA, USA, May 7-9, 2015*, Yoshua Bengio and Yann LeCun. http://arxiv.org/abs/1409.0473.

[4] 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.

[5] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. In *Noise Reduction in Speech Processing*. Springer, 1–4.

[6] Raymond P. L. Buse and Westley R. Weimer. 2010. Automatically documenting program changes. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. 33–42.

[7] Qiuyuan Chen, Xin Xia, Han Hu, David Lo, and Shanping Li. 2021. Why my code summarization model does not work: Code comment improvement with category prediction. *ACM Transactions on Software Engineering and Methodology* 30, 2 (2021), 1–29.

[8] Luis Fernando Cortés-Coy, Mario Linares-Vásquez, Jairo Aponte, and Denys Poshyvanyk. 2014. On automatically generating commit messages via summarization of source code changes. In *Proceedings of the 2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. IEEE, 275–284.

[9] Deborah Coughlin. 2003. Correlating automated and human assessments of machine translation quality. In *Proceedings of MT summit IX*. Citeseer, 63–70.

[10] Michael Denkowski and Alon Lavie. 2014. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the EACL 2014 Workshop on Statistical Machine Translation*.

[11] 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 *Proceedings of the 2013 35th International Conference on Software Engineering*. IEEE, 422–431.

[12] Joseph L. Fleiss, Bruce Levin, and Myunghee Cho Paik. 2013. *Statistical Methods for Rates and Proportions*. John Wiley & Sons.

[13] Andrew Forward and Timothy C. Lethbridge. 2002. The relevance of software documentation, tools and technologies: A survey. In *Proceedings of the 2002 ACM Symposium on Document Engineering*. 26–33.

[14] Yvette Graham, Timothy Baldwin, Alistair Moffat, and Justin Zobel. 2013. Continuous measurement scales in human evaluation of machine translation. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*. 33–41.

[15] David Gros, Hariharan Sezhiyan, Prem Devanbu, and Zhou Yu. 2020. Code to comment" Translation": Data, metrics, baselining & evaluation. In *(ASE'20)*. Association for Computing Machinery, 746–757. DOI: 10.1145/3324884.3416546

[16] Sonia Haiduc, Jairo Aponte, and Andrian Marcus. 2010. Supporting program comprehension with source code summarization. In *Proceedings of the 2010 ACM/IEEE 32nd International Conference on Software Engineering*. IEEE, 223–226.

[17] Sonia Haiduc, Jairo Aponte, Laura Moreno, and Andrian Marcus. 2010. On the use of automated text summarization techniques for summarizing source code. In *Proceedings of the 2010 17th Working Conference on Reverse Engineering*. IEEE, 35–44.

[18] Dennis E. Hinkle, William Wiersma, and Stephen G. Jurs. 2003. *Applied Statistics for the Behavioral Sciences*. Vol. 663. Houghton Mifflin College Division.

[19] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *Proceedings of the 2018 IEEE/ACM 26th International Conference on Program Comprehension*. IEEE, 200–210.

[20] 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.

[21] Yuan Huang, Qiaoyang Zheng, Xiangping Chen, Yingfei Xiong, Zhiyong Liu, and Xiaonan Luo. 2017. Mining version control system for automatically generating commit comment. In *Proceedings of the 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 414–423.

[22] 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*. 2073–2083.

[23] Siyuan Jiang, Ameer Armaly, and Collin McMillan. 2017. Automatically generating commit messages from diffs using neural machine translation. In *Proceedings of the 2017 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 135–146.

[24] Alexander LeClair, Sakib Haque, Lingfei Wu, and Collin McMillan. 2020. Improved code summarization via a graph neural network. In *Proceedings of the 28th International Conference on Program Comprehension*. 184–195.

[25] Alexander LeClair, Siyuan Jiang, and Collin McMillan. 2019. A neural model for generating natural language summaries of program subroutines. In *Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering*. IEEE, 795–806.

[26] T. C. Lethbridge, J. Singer, and A. Forward. 2003. How software engineers use documentation: The state of the practice. *IEEE Software* 20, 6 (2003), 35–39.

[27] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Proceedings of the Text Summarization Branches Out*. 74–81.

[28] Mario Linares-Vásquez, Luis Fernando Cortés-Coy, Jairo Aponte, and Denys Poshyvanyk. 2015. Changescribe: A tool for automatically generating commit messages. In *Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, 709–712.

[29] Chia-Wei Liu, Ryan Lowe, Iulian Vlad Serban, Mike Noseworthy, Laurent Charlin, and Joelle Pineau. 2016. How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2122–2132.

[30] Qin Liu, Zihe Liu, Hongming Zhu, Hongfei Fan, Bowen Du, and Yu Qian. 2019. Generating commit messages from diffs using pointer-generator network. In *Proceedings of the 2019 IEEE/ACM 16th International Conference on Mining Software Repositories*. IEEE, 299–309.

[31] Shangqing Liu, Yu Chen, Xiaofei Xie, Jing Kai Siow, and Yang Liu. 2020. Retrieval-augmented generation for code summarization via hybrid GNN. In *Proceedings of the International Conference on Learning Representations*.

[32] 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.

[33] Pablo Loyola, Edison Marrese-Taylor, Jorge Balazs, Yutaka Matsuo, and Fumiko Satoh. 2018. Content aware source code change description generation. In *Proceedings of the 11th International Conference on Natural Language Generation*. 119–128.

[34] Pablo Loyola, Edison Marrese-Taylor, and Yutaka Matsuo. 2017. A neural architecture for generating natural language descriptions from source code changes. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, 287–292. https://aclanthology.org/P17-2045.

[35] Paul W. McBurney and Collin McMillan. 2014. Automatic documentation generation via source code summarization of method context. In *Proceedings of the 22nd International Conference on Program Comprehension*. 279–290.

[36] 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 *Proceedings of the 2013 21st International Conference on Program Comprehension*. IEEE, 23–32.

[37] Laura Moreno, Andrian Marcus, Lori Pollock, and K. Vijay-Shanker. 2013. Jsummarizer: An automatic generator of natural language summaries for Java classes. In *Proceedings of the 2013 21st International Conference on Program Comprehension*. IEEE, 230–232.

[38] Jekaterina Novikova, Ondřej Dušek, Amanda Cercas Curry, and Verena Rieser. 2017. Why we need new evaluation metrics for NLG. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2241–2252.

[39] 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.

[40] Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 379–389. DOI: 10.18653/v1/D15-1044

[41] Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. 1073–1083.

[42] Shikhar Sharma, Layla El Asri, Hannes Schulz, and Jeremie Zumer. 2017. Relevance of unsupervised metrics in task-oriented dialogue for evaluating natural language generation. *CoRR* abs/1706.09799. http://arxiv.org/abs/1706.09799.

[43] Xiaotao Song, Hailong Sun, Xu Wang, and Jiafei Yan. 2019. A survey of automatic generation of source code comments: Algorithms and techniques. *IEEE Access* 7 (2019), 111411–111428.

[44] 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.

[45] Sean Stapleton, Yashmeet Gambhir, Alexander LeClair, Zachary Eberhart, Westley Weimer, Kevin Leach, and Yu Huang. 2020. A human study of comprehension and code summarization. In *Proceedings of the 28th International Conference on Program Comprehension*. 2–13.

[46] Amanda Stent, Matthew Marge, and Mohit Singhai. 2005. Evaluating evaluation methods for generation in the presence of variation. In *Proceedings of the International Conference on Intelligent Text Processing and Computational Linguistics*. Springer, 341–351.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the Advances in Neural Information Processing Systems*. 5998–6008.

[48] 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.

[49] Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, and Philip S. Yu. 2018. Improving automatic source code summarization via deep reinforcement learning. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 397–407.

[50] Bolin Wei, Yongmin Li, Ge Li, Xin Xia, and Zhi Jin. 2019. Retrieve and refine: exemplar-based neural comment generation. In *Proceedings of the 35th ACM/IEEE International Conference on Automated Software Engineering*.

[51] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li. 2018. Measuring program comprehension: A large-scale field study with professionals. *IEEE Transactions on Software Engineering* 44, 10 (2018), 951–976.

[52] Shengbin Xu, Yuan Yao, Feng Xu, Tianxiao Gu, Hanghang Tong, and Jian Lu. 2019. Commit message generation for source code changes. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 3975–3981.

[53] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, and Xudong Liu. 2020. Retrieval-based neural source code summarization. In *Proceedings of the 2020 IEEE/ACM 42nd International Conference on Software Engineering*. IEEE, 1385–1397.

Received January 2021; revised September 2021; accepted November 2021