Using GPT-4 for Source Code Documentation

Magdalena Kneidinger, Markus Feneberger, Reinhold Plösch
Institute of Business Informatics - Software Engineering
Johannes Kepler University
Linz, Austria
k12019988@students.jku.at, {markus.feneberger,reinhold.ploesch}@jku.at

***Abstract*—Writing good software documentation imposes significant effort. Large Language Models (LLMs) could potentially streamline that process, though. So, the question arises whether current LLMs are able to generate valid code documentation for classes and methods on basis of the bare code. According to literature, various such models have the capability to generate documentation that is on par with or even superior to reference documentation. In our experimental study using zero-shot prompting, we found that the model GPT-4 by OpenAI leads to poor results when measuring similarity to the reference documentation on class level. Thus, GPT-4 is not yet usable for generating class documentation. On method level, however, the model achieved higher similarity ratings and can be considered applicable for the use case.**

***Index Terms*—LLM, Large Language Model, GPT-4, Software Documentation, Class Documentation, Method Documentation**

I. Introduction

The documentation of software is of crucial importance in the software development process. It contributes significantly to the comprehensibility, maintainability and further development of software projects [1]. However, the creation of high-quality documentation tends to involve considerable effort. This often means that documentation is neglected or does not achieve the desired level of quality [2]. In recent years, large language models (LLMs) have emerged as promising technologies. The application *ChatGPT*, based on OpenAI's GPT model series, in particular has attracted a lot of attention [3]. These models are able to generate human-like text and respond to requests in natural language [3]. In addition, some LLMs are able to handle source code, which is attracting increasing interest for use in software development tasks including the generation of software documentation [4]–[6].

A. Motivation & Objective

Many programming languages (or, their tooling, actually) offer a way to document code through structured comments. Examples for this include Java's *Javadoc* or JavaScript's *JSDoc* comment formats, or Python's *Docstring* system which utilises string literals. Poor documentation is common in industry and little effort is put in improving it [7]. With the current speed of developing newer LLMs, automation of code documentation might be on its way, too. This lead us to the following research question: What results does Java class and method *documentation generated by GPT-4* achieve, compared to manually written *reference documentation*?

To answer this question, we investigated GPT-4's capability to generate class and method documentation for Java code, i.e., Javadoc comments for classes and methods, in an experiment. Other large language models, including previous versions of the GPT model, have already been surveyed for this use case.

B. Related Work

Table I presents an overview of recent work on code documentation generation with LLMs. It can be seen that the automated generation of method level documentation has already been investigated rather extensively. Concerning the class level, however, scientific literature is currently lacking.

II. Research Method

To answer the asked research question, we conducted an experiment on the quality of GPT-generated source code documentation. In this section, the experiment setup is explained in detail, including a description of the used model and the used Java classes. This is followed by the created prompt designs and an overview of the applied evaluation methods.

A. Datasets

As test data, we used eleven Java classes to have the AI model generate Javadoc comments for class level, and method level, for a selection of five methods per class. This leads to a total of eleven classes and 55 methods, including seven classes from the Java Development Kit (JDK), one from the EJML[1] package and three from the JHotDraw[2] package.

B. Applied LLM

OpenAI's latest model, GPT-4, has come out in March 2023 [14]. As per Table I and how new the model still is, few research papers on it have been published yet. For that reason, we used GPT-4 in this experiment, to address this research gap and investigate the model's effectiveness in terms of code documentation. GPT-4 is able to process visual as well as textual input; it outperforms many existing LLMs in a number of NLP tasks and eclipses the vast majority of SOTA models [14].

C. Prompt Design

To test the basic performance of GPT-4, we chose a zero-shot prompting approach, consisting of two prompts: one for the method-level and one for the class-level documentation. Thus, the same prompts have been used for all test data for

[1]http://ejml.org
[2]https://www.randelshofer.ch/oop/jhotdraw/

TABLE I
PREVIOUS STUDIES ON LLM-GENERATED CODE DOCUMENTATION

Citation	Model(s)	Language(s) (Dataset(s))	Layer(s)	Evaluation
[3]	ChatGPT (GPT 3.5), compared with CodeBERT & CodeT5	Python (CSN)	Method	BLEU, METEOR, ROUGE-L
[8]	GPT-3.5, GPT-4, Bard, Llama2, Starchat	Python	Inline, Method, Package	Manual
[9]	GPT-3.5	Java (Funcom)	Method	Manual
[10]	CodeX, compared with CodeBERT variants, CodeT5, few-shot prompting focus	Various[3](CodeXGLUE)	Method	Smoothed BLEU-4
[11]	CodeX Few-shot prompting and ICL[4]focus	Java (Funcom, TLC)	Method	BLEU, ROUGE-L, METEOR, Manual
[12]	CodeBERT, GraphCodeBERT	Java, Python	-	BLEU-4
[6], [13]	CodeT5+, compared with RoBERTa, CodeBERT, UniXcoder, PLBART, CodeT5	Various[1] (CodeSearch-Net)	Method	Smoothed BLEU-4

both documentation layers, to ensure comparability between test classes and methods. After a few trial runs, the prompts have been constructed as follows:

- System Prompt: The model has been assigned the role of a helpful assistant for Javadoc code summarising.
- Task: The model has been given the task with clear instructions. Three quality criteria were mentioned: Correctness, Completeness and Conciseness as well as the code level to generate documentation for.
- Datasets: The whole, uncommented Java class was given to the model.
- Output: The model was instructed to return the Java code with the newly inserted Javadoc comments.

Taking all these aspects into account, we went with these two prompts:

- Add accurate, complete and concise Javadoc-documentation to the class and all methods and fields of this java-class: {javaCode}. The output should contain the whole source code of the given java-class with all generated Javadoc-documentation.
- Add accurate, complete and concise Javadoc-documentation to the following methods of this java-class: {javaCode}. The methods are: [Names of the five selected methods]; please also consider the annotations. The output should contain the code of these java-methods with the generated javadoc-documentation.

D. Evaluation

The quality of the generated code documentation was evaluated based on different metrics. As automated metrics, Smoothed-BLEU-4 and ROUGE-1 have been used. Both metrics are commonly used for similar experiments [10], [11], [13], [15], see Table I. They both measure the similarity of two texts [16], [17]. The improved version of BLEU, *Smoothed-BLEU*, attempts to improve on some problems as the original only correlates weakly with human judgement.

In addition to BLEU and ROUGE, we performed a manual evaluation as well. Since the two metrics only evaluate the lexical discrepancy between the generated and the reference documentation, they are not sufficient to detect semantic differences [11]. Furthermore, we cannot assume that the original documentation is perfect, an aspect that has to be considered in the comparison of generated documentation. For that reason, the generated documentation was checked for correctness, completeness and conciseness, which are three common quality criteria [9]. To allow quantitative evaluation, we defined a four-part scale for each criterion:

III. Results

In this section, the results of the conducted experiment are presented, grouped by documentation level and type of evaluation.

A. Class-level documentation

Figure 1 shows the results of the manual evaluation of the class documentation. The points have been averaged across all classes, by criterion. The bar chart presents the evaluation points of the reference documentation (light) and the generated documentation (dark) pairwise.

Considering **correctness**, the diagram shows that both the reference documentation and the generated documentation have reached the full three points, i.e., do not include any false information. This is not the case for **completeness**, where the generated documentation has performed significantly worse at only 1.3 pts. Overall, GPT-4 tended to generate shorter, more general descriptions that miss key elements that are included in the reference documentation. This also leads to GPT-4 almost matching the reference documentation in **conciseness**, though.

Applying the metrics ROUGE-1 and Smoothed-BLEU-4 showed that GPT-4 did not generate class documentation that is similar to the reference documentation:

- ROUGE-1: 17 %
- Smoothed-BLEU-4: 3.38 %

The above-mentioned shorter comments from GPT are partly responsible for this. On five classes (i.e., almost half of them), the BLEU result was less than 0.1 per cent. The highest BLEU score was achieved on `java.lang.Integer` at 16.6 per cent. While being generally higher, the ROUGE score correlated strongly (Pearson correlation coefficient of 0.9433). The

[3]Six languages, including Java and Python
[4]In-context Learning

TABLE II
CRITERIA OF THE MANUAL EVALUATION

Criterion	Points	Definition
Correctness	0	The documentation is incorrect as a whole or missing entirely
	1	The documentation contains more than one piece of false information. What is false?
	2	The documentation contains one piece of false information. What is false?
	3	The documentation is correct.
Completeness	0	The documentation is missing entirely.
	1	The documentation lacks multiple important pieces of information. What is missing?
	2	The documentation is missing one piece of important information. What is missing?
	3	The documentation is complete.
Conciseness	0	The documentation only contains unnecessary information or is missing entirely.
	1	The documentation contains more than one unnecessary piece of information. What is unnecessary?
	2	The documentation contains one unnecessary piece of information. What is unnecessary?
	3	The documentation is concise and hence contains no unnecessary information.

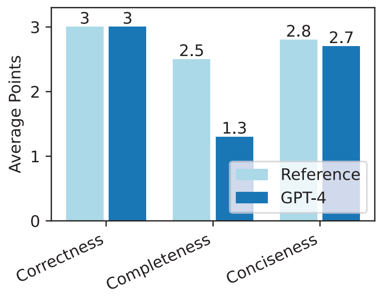

Fig. 1. Average Results on Class level

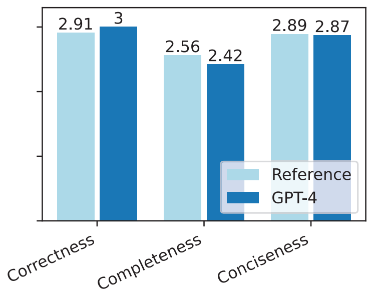

Fig. 2. Average Results on Method level

correlation of both scores with the manual evaluation is much weaker, as to be expected from the raw data: ROUGE and Manual correlate at 0.0953, BLEU and Manual at 0.1592 per cent. Both indicate weak, but not reliable, positive correlation [see 18].

B. Method-level documentation

The results of the manual evaluation of the method documentation are shown in Figure 2, in the same way as the class results in Figure 1. It can be seen that, unlike at the class level, GPT-4 (3 pts.) has managed to surpass the reference documentation (2.91 pts.) quality in correctness. Also in the two other criteria, the model was able to almost match it. This means that GPT-4 produced (significantly) more complete method documentation than class documentation. On several methods, GPT-4 has also outperformed the reference documentation in completeness.

Bearing the previously learned correlation in mind, the generated documentation can also be expected to be more similar (i.e., higher BLEU and ROUGE scores) to the reference documentation. This is indeed the case:

- ROUGE-1: 42.47 %
- Smoothed-BLEU-4: 18.38 %

The Pearson correlation coefficient was also calculated for the method data:

- ROUGE with BLEU: 0.9112
- ROUGE with Manual: 0.1938
- BLEU with Manual: 0.2075

So, both similarity metrics strongly correlate among the method documentation as well. The correlation of each metric with the manual evaluation results is stronger than at the class level but still quite weak. Thus, a higher similarity indicates a slightly higher probability that a generated Javadoc method comment is correct, complete and concise. The major *threat to validity* is besides the limited number of investigated documentation the fact that the quality of the reference documentation and of the generated documentation was judged by two experts, only. Nevertheless, we think that only using similarity metrics is not convincing, but needs expert judgement from point of view of software developers.

IV. CONCLUSIONS

To evaluate whether the large language model GPT-4 is capable of generating Javadoc comments for classes and methods, we conducted a two-part experiment.

At class level, both the generated and the reference documentation received maximum scores in terms of accuracy. However, the generated documentation showed significant deficits in completeness compared to the reference documentation, as they were shorter and less detailed and often lacked important details and standard Javadoc tags. In terms of conciseness, the generated and reference documentation scored similarly. In general, the generated documentation could not match the quality of the reference documentation. On two of the eleven classes, however, it could be considered on par.

At the method level, GPT-4 performed better than the reference documentation in terms of correctness, as GPT-4 did not generate any incorrect information, as was the case at the class level. However, the generated documentation had a slightly lower completeness score, particularly due to missing '`@see`' tags and references to special cases or IEEE standards. Conversely, the reference documentation often lacked descriptions of return and parameter values. The ratings for the conciseness of the method documentation were almost identical, which indicates that both the generated and the reference documentation largely contained only relevant information. The aggregated results across all classes show that in most cases the reference documentations scored slightly higher than the generated documentations, but without significant differences, indicating an effective performance of GPT-4.

Since there is no evaluation of class-level documentation in literature yet, we could not make direct comparisons. Nevertheless, higher-level (such as class-level) documentation imposes a challenge for large language models [19] in general. Concerning the method level, though, our results fit with previous insights in that the generated documentation almost matches the reference documentation in quality. Other studies have even found GPT-4 producing *better* method documentation than the reference documentation [8], but we do not know about the quality of the latter. GPT mostly relying on the control flow instead of method and variable names [3] is a conclusion that cannot be drawn from our results [see 3], [12]. We currently conduct a study on the migration of a larger hardly documented software system where we will use GPT-4 for generating documentation - in an attempt to have a better input basis for code transformation, unit test enhancements, etc.

References

[1] A. Y. Wang, D. Wang, J. Drozdal, *et al.*, "Documentation Matters: Human-Centered AI System to Assist Data Science Code Documentation in Computational Notebooks," en, *ACM Transactions on Computer-Human Interaction*, vol. 29, no. 2, pp. 1–33, Apr. 2022, ISSN: 1073-0516, 1557-7325. DOI: 10.1145/3489465.

[2] R. S. Geiger, N. Varoquaux, C. Mazel-Cabasse, and C. Holdgraf, "The Types, Roles, and Practices of Documentation in Data Analytics Open Source Software Libraries: A Collaborative Ethnography of Documentation Work," en, *Computer Supported Cooperative Work (CSCW)*, vol. 27, no. 3-6, pp. 767–802, Dec. 2018, ISSN: 0925-9724, 1573-7551. DOI: 10.1007/s10606-018-9333-1.

[3] W. Sun, C. Fang, Y. You, *et al.*, *Automatic Code Summarization via ChatGPT: How Far Are We?* arXiv:2305.12865 [cs], May 2023. DOI: 10.48550/arXiv.2305.12865.

[4] J. Cao, M. Li, M. Wen, and S.-c. Cheung, "A study on Prompt Design, Advantages and Limitations of ChatGPT for Deep Learning Program Repair," 2023. DOI: 10.48550/ARXIV.2304.08191.

[5] Z. Feng, D. Guo, D. Tang, *et al.*, "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," en, in *Findings of the Association for Computational Linguistics: EMNLP 2020*, Online: Association for Computational Linguistics, 2020, pp. 1536–1547. DOI: 10.18653/v1/2020.findings-emnlp.139.

[6] Y. Wang, H. Le, A. D. Gotmare, N. D. Q. Bui, J. Li, and S. C. H. Hoi, *CodeT5+: Open Code Large Language Models for Code Understanding and Generation*, arXiv:2305.07922 [cs], May 2023.

[7] M. Kajko-Mattsson, "A Survey of Documentation Practice within Corrective Maintenance," en, *Empirical Software Engineering*, vol. 10, no. 1, pp. 31–55, Jan. 2005, ISSN: 1382-3256. DOI: 10.1023/B:LIDA.0000048322.42751.ca.

[8] S. S. Dvivedi, V. Vijay, S. L. R. Pujari, S. Lodh, and D. Kumar, *A Comparative Analysis of Large Language Models for Code Documentation Generation*, arXiv:2312.10349 [cs], Dec. 2023.

[9] C.-Y. Su and C. McMillan, *Distilled GPT for Source Code Summarization*, arXiv:2308.14731 [cs], Aug. 2023. DOI: 10.48550/arXiv.2308.14731.

[10] T. Ahmed and P. Devanbu, "Few-shot training LLMs for project-specific code-summarization," en, in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, Rochester MI USA: ACM, Oct. 2022, pp. 1–5, ISBN: 978-1-4503-9475-8. DOI: 10.1145/3551349.3559555.

[11] M. Geng, S. Wang, D. Dong, *et al.*, "Large Language Models are Few-Shot Summarizers: Multi-Intent Comment Generation via In-Context Learning," 2023. DOI: 10.48550/ARXIV.2304.11384.

[12] A. H. Mohammadkhani, C. Tantithamthavorn, and H. Hemmati, *Explainable AI for Pre-Trained Code Models: What Do They Learn? When They Do Not Work?* arXiv:2211.12821 [cs], Aug. 2023.

[13] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation," en, in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, 2021, pp. 8696–8708. DOI: 10.18653/v1/2021.emnlp-main.685.

[14] OpenAI, J. Achiam, S. Adler, *et al.*, *GPT-4 Technical Report*, arXiv:2303.08774 [cs], Mar. 2024.

[15] T. Kajiura, N. Souma, M. Sato, M. Takahashi, and K. Kuramitsu, "An additional approach to pre-trained code model with multilingual natural languages," in *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*, ISSN: 2640-0715, Dec. 2022, pp. 580–581. DOI: 10.1109/APSEC57359.2022.00090.

[16] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A method for automatic evaluation of machine translation," en, in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, Philadelphia, Pennsylvania: Association for Computational Linguistics, 2001, p. 311. DOI: 10.3115/1073083.1073135.

[17] C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," in *Text Summarization Branches Out*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81.

[18] E. Reiter, "A Structured Review of the Validity of BLEU," en, *Computational Linguistics*, vol. 44, no. 3, pp. 393–401, Sep. 2018, ISSN: 0891-2017, 1530-9312. DOI: 10.1162/coli_a_00322.

[19] S. A. Rukmono, L. Ochoa, and M. R. Chaudron, "Achieving High-Level Software Component Summarization via Hierarchical Chain-of-Thought Prompting and Static Code Analysis," in *2023 IEEE International Conference on Data and Software Engineering (ICoDSE)*, Toba, Indonesia: IEEE, Sep. 2023, pp. 7–12, ISBN: 9798350381382. DOI: 10.1109/ICoDSE59534.2023.10292037.