Report on ChatGPTs ability to generate code documentation/comments

"A Comparative Analysis of Large Language Models for Code Documentation Generation"	
Aspect	**Details**
Idea	Evaluates the effectiveness of LLMs in generating code documentation, comparing models like GPT-3.5, GPT-4, Bard, Llama2, and StarChat across different documentation levels (inline, function, and file-level).
Method	Uses a checklist-based evaluation approach. LLMs generate documentation using standardized prompts, and two expert reviewers assess the output on six key metrics. Statistical analysis (ANOVA) is applied to determine significance.
Dataset	14 Python code snippets from well-documented open-source projects (Bitcoin, TensorFlow, SQLAlchemy, Scikit-Learn, and Reddit). The dataset includes inline, function-level, and file-level documentation cases.
Evaluation Metrics	- **Accuracy** (factual correctness) - **Completeness** (coverage of code aspects) - **Relevance** (alignment with the code) - **Understandability** (clarity for users) - **Readability** (formatting and structure) - **Time Taken** (efficiency of generation)
Results	- GPT-3.5, GPT-4, and Bard outperform human-generated documentation. - GPT-4 delivers the best quality but takes the longest time. - Open-source models (Llama2, StarChat) perform worse, with StarChat being the weakest. - File-level documentation is the least effective, while inline and function-level documentation perform significantly better.
Reflection	The paper provides a strong comparative analysis of LLMs for code documentation, showing that closed-source models outperform open-source ones. Its objective evaluation method is a key contribution, but limitations include a small dataset and challenges with file-level documentation. Future research should focus on improving summarization, expanding datasets, and integrating LLMs into real-world development tools.

"Automatic Code Summarization via ChatGPT: How Far Are We?."	
Aspect	**Details**
Idea	The research investigates ChatGPT's effectiveness in automatic code summarization by comparing its performance against state-of-the-art (SOTA) models.
Method	The authors designed prompts to guide ChatGPT in generating comments for code snippets from the CSN-Python test set. They then compared these summaries with those produced by SOTA models: NCS, CodeBERT, and CodeT5.
Dataset	The study utilized the CSN-Python dataset, a widely-used collection of Python code snippets and their corresponding summaries.
Evaluation Metrics	Performance was assessed using three metrics: BLEU, METEOR, and ROUGE-L, which measure the quality of generated text against reference summaries.
Results	ChatGPT's performance was significantly lower than the SOTA models in terms of BLEU and ROUGE-L scores. This indicates that ChatGPT's code summaries were less aligned with reference summaries compared to those produced by the other models..
Reflection	The findings suggest that while ChatGPT has potential in code summarization, it currently lags behind specialized models. The study highlights the need for further research to enhance ChatGPT's capabilities in this domain.

"Can Developers Prompt? A Controlled Experiment for Code Documentation Generation."	
Aspect	**Details**
Idea	This study investigates how well developers, both students and professionals, can prompt LLMs to generate code documentation and compares ad-hoc prompts with predefined few-shot prompts.
Method	A controlled, between-subject experiment was conducted where participants documented two Python functions. The ad-hoc group crafted their prompts, while the control group used a predefined few-shot prompt in VS Code. The study included quantitative (statistical tests) and qualitative analysis based on human evaluations.
Dataset	Two Python functions from the Karabo open-source project were selected for the tasks. These functions varied in complexity to test different levels of comprehension and documentation generation capabilities.
Evaluation Metrics	The generated documentation was assessed based on six quality dimensions: grammatical correctness, readability, missing information, unnecessary information, usefulness, and helpfulness.
Results	Predefined prompts generally produced higher-quality documentation in terms of readability and conciseness. Professionals performed better with ad-hoc prompts than students, often by using documentation-specific keywords. Both tools improved code comprehension, particularly for more complex functions.
Reflection	The study highlights the gap in developers' prompt engineering skills. While predefined prompts offer better initial results, documentation is often refined iteratively. Future research could focus on prompt templates, iterative documentation support, and human-centered evaluation metrics.

"Using GPT-4 for Source Code Documentation."	
Aspect	**Details**
Idea	Investigates GPT-4's effectiveness in generating Java class-level and method-level documentation compared to manually written reference documentation.
Method	The study uses zero-shot prompting. GPT-4 generates Javadoc comments for both class and method levels using predefined prompts. The output is evaluated using both automated (BLEU, ROUGE) and manual methods.
Dataset	Eleven Java classes and 55 methods were selected from open-source projects, including the Java Development Kit (JDK), EJML, and JHotDraw.
Evaluation Metrics	• Correctness (accuracy of information) • Completeness (coverage of all necessary details) • Conciseness (avoiding unnecessary information) • BLEU (text similarity) • ROUGE-1 (text overlap)
Results	• GPT-4 documentation for methods was more accurate than the reference but slightly less complete. • At the class level, GPT-4 performed poorly in completeness, though its correctness was high. • Method-level BLEU and ROUGE scores were significantly higher than those for class-level documentation.
Reflection	GPT-4 is effective for method-level documentation but struggles with class-level documentation due to missing details and Javadoc tags. The study suggests improving LLM prompting strategies and leveraging expert evaluation for better quality.

"The Role of ChatGPT in Computer programing."	
Aspect	**Details**
Idea	The research explores the role and capabilities of ChatGPT in computer programming. It focuses on how ChatGPT can assist developers by offering features such as code completion, error correction, optimization, documentation generation, chatbot development, and text-to-code generation. The study emphasizes ChatGPT's potential to improve developer efficiency and accuracy.
Method	The study investigates ChatGPT's programming-related capabilities by analyzing its ability to predict code snippets, fix syntax errors, suggest optimizations, generate documentation, and assist in chatbot development. It provides various examples demonstrating how ChatGPT can enhance coding processes and improve developer workflows.
Dataset	The research does not specify a particular dataset. Instead, it relies on ChatGPT's pre-trained knowledge, which comes from diverse programming-related texts, documentation, and code repositories. The examples provided illustrate ChatGPT's ability to generate and refine code across multiple programming languages.
Evaluation Metrics	No formal evaluation metrics are mentioned in the research. However, the effectiveness of ChatGPT is assessed qualitatively through examples demonstrating its accuracy in code suggestions, error correction, and documentation generation. The paper highlights improvements in productivity, code quality, and efficiency as key success indicators.
Results	The study concludes that ChatGPT is a valuable tool for developers, helping them reduce coding errors, improve code quality, and accelerate development processes. It demonstrates ChatGPT's effectiveness in providing real-time coding assistance, automating documentation, and refining code structure. However, it acknowledges that ChatGPT still requires human oversight for complex tasks.
Reflection	The research highlights the benefits of ChatGPT in programming while recognizing its limitations. While it enhances productivity and supports developers in various tasks, it is not a replacement for human expertise, particularly in handling complex and context-specific problems. The study suggests that ChatGPT is best used as an assistive tool rather than a standalone solution for software development.

References

A Comparative Analysis of Large Language Models for Code Documentation Generation.

[1] 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,” Proc. 1st ACM Int. Conf. AIware, Porto de Galinhas, Brazil, July 2024.

Automatic Code Summarization via ChatGPT: How Far Are We?

Study made and published by:
Weisong Sun1,2, Chunrong Fang1*, Yudu You1, Yun Miao1, Yi Liu2, Yuekang Li3, Gelei Deng2, Shenghan Huang1, Yuchen Chen1, Quanjun Zhang1, Hanwei Qian1, Yang Liu2, Zhenyu Chen1 1State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China 2Nanyang Technological University, Singapore 3 University of New South Wales, Australia

Can Developers Prompt? A Controlled Experiment for Code Documentation Generation

[1] E. Aghajani, C. Nagy, O. L. Vega-Marquez, M. Linares-V ´ asquez, ´ L. Moreno, G. Bavota, and M. Lanza, “Software documentation issues unveiled,” in 2019 IEEE/ACM 41st International Conference on Software Engineering. New York, NY, USA: IEEE, 2019, pp. 1199–1210.

Using GPT-4 for Source Code Documentation

[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.

The Role of ChatGPT in computer programming

1] T. Phillips et al., “Exploring the use of GPT-3 as a tool for evaluating text-based collaborative discourse,” Companion Proc. 12th, 2022, p. 54.[2] B. Lund and T. Wang, “Chatting about ChatGPT: How May AI and GPT Impact Academia and Libraries?,” Lund, B. D., & Wang, 2023.[3] P. H. Winston, Artificial Intelligence. Boston, MA, USA: Addison-Wesley Longman Publ. Co., Inc., 1984.[4] E. Charniak et al., Artificial Intelligence Programming. New York, NY, USA: Psychology Press, 2014.[5] S. L. Tanimoto, The Elements of Artificial Intelligence: An Introduction Using LISP. Rockville, MD, USA: Computer Science Press, Inc., 1987.