Physical Education, Health and Social Sciences

https://journal-of-social-education.org

E-ISSN: 2958-5996
P-ISSN: 2958-5988

Using NLP and AI to Enhance Software Documentation and Code Comprehension

Abdulmalik Ibrahim [1], Muhammad Baryal [2], Asad Ullah [2], Muhammad Shoaib[3], Muhammad Ghayas Khan[4]

[1] School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, Scotland Email: ibrahimmalik85@gmail.com
[2] Department of Computer Science, Kohat University of Science and Technology (KUST) (Hangu Campus), Pakistan Email: baryalkhan2060@gmail.com, asadbangash2060@gmail.com
[3] Department of Computer Science, University of Haripur, Email: shoaibnazir944@gmail.com
[4] Department of Business Administration, International Islamic University Islamabad (IIUI) Email: ghayas1012@gmail.com

DOI: https://doi.org/10.63163/jpehss.v3i2.292

Abstract

Software documentation plays a critical role in code comprehension, maintenance, and collaboration, yet it is often incomplete, outdated, or inconsistently written. This study explores the application of Artificial Intelligence (AI) and Natural Language Processing (NLP) techniques to automatically generate accurate and context-aware documentation for software code. Leveraging transformer-based models such as CodeT5, GraphCodeBERT, and GPT-3, we developed and evaluated a system capable of producing meaningful summaries of code functions and classes. A comparative analysis between human-written and AI-generated documentation was conducted using both quantitative metrics (BLEU, ROUGE, F1) and qualitative feedback from professional developers. The results indicate that AI-generated documentation significantly improves code readability and developer efficiency, reducing comprehension time and enhancing accuracy in understanding complex code. Additionally, real-time integration of the tool within development environments proved beneficial for continuous documentation support. While AI still faces challenges in handling domain-specific code and interpreting poorly written segments, the overall impact on documentation quality is substantial. This research underscores the potential of NLP-driven tools to automate and standardize documentation practices, offering a scalable solution to one of software engineering's longstanding challenges. Future work aims to integrate context-awareness, multilingual support, and interactive querying features to further enhance developer experience.

Introduction

An indispensable trait for documentation is to be lucid, constructive, and abreast of the software development world, which keeps evolving like a chameleon. Documentation for any software acts as a critical bridge between human considerations and machine considerations in programming; it explains, justifies, and shows how the code is structured, used, modified, or extended[1]. With its essential nature being heralded, software documentation is still invariably one of the most often neglected aspects of the development process. Due to the constant pressure of deadlines, often changing project requirements, or plain preference of code over documentation, developers find it hard to maintain up-to-date documentation. Consequently, the absence of documentation breeds

misunderstandings, maintenance problems, and extended time for a new team member to onboard[2]. In this situation, the amalgamation of AI and NLP provides a unique opportunity to transform the way documentation is conceived, edited, and understood. AI in particular, based on machine learning and deep learning, continues to make possibilities in several realms by carrying out tasks that traditionally would require human intelligence[3]. Natural Language Processing, a subfield of AI, deals with the effective use of machines to understand, interpret and manipulate human languages. The use of AI and NLP techniques along with software engineering principles has become a focus in the area of enhancing various activities in the software lifecycle. One of the most interesting application domains is in the area of software documentation and code understanding by means of automation and enrichment. The intelligent algorithms help AI to analyze source code, understand its semantics, and generate human-comprehensible explanations and insert relevant documentation snippets [4]. Code and their understanding are very much important for developers, especially booting for debugging, code reviews, and feature extension. The tradition-credentialled tools for understanding the codes directly do static analysis and syntax highlighting approaches, offering little understanding of the actual behavior or purpose of the codes[5]. Code and their understanding are very much important for developers, especially booting for debugging, code reviews, and feature extension. The tradition-credentialled tools for understanding the codes directly do static analysis and syntax highlighting approaches, offering little understanding of the actual behavior or purpose of the codes [6]. More and more recent developments in transformer-based models such as BERT, GPT, and CodeBERT have considerably improved the capacities of NLP when dealing with programming languages. Such models are trained using large-scale repositories of code. It has shown outstanding results in various forms of tasks, such as code summarization, comment generation, and documentation synthesis[7]. The inclusion of such models into integrated developmental environments (IDEs) and code-hosting platforms, such as GitHub, gives software developers the chance to use intelligent assistants to suggest documentation, auto-generate function descriptions, generate code comments in multiple languages, and so on. Such automation greatly increases the quality of documentation and saves a lot of time and effort for development teams[8]. The NLP and AI-infused documentation process is valuable for collaborative development environments, in which multiple programmers work on the same project. Given that most often diverse developers would be involved, some may not comprehend the reasoning behind some of those code decisions. In this case, AI-based documentation would assist with contextual information, giving version-based explanations, and allow tracking of any alterations for making code transparent, preventing any level of knowledge being siloed[9]. Similarly, these tools can catch documentation-actual code mismatches to make sure information and thereby standardize the software system's reliability. One of the important factors for this study will be the ever-increasing complexities of modern-day software applications. The increasing modularization and decentralization of software architectures that are promoted nowadays by microservices, cloud computing, and continuous integrations are making these systems complex[10]. Without appropriate documentation, understanding relationships in these systems is a very complex task. This is where AI and NLP can really help with automatic interaction mapping, generation of inter-relationship architecture diagrams, and description of responsibilities and dependencies of each of the modules concerne. AI is promising in the sphere of software documentation but comes with its own set of challenges. The foremost issue is concerning the accuracy and reliability of AI-generated content. While models like CodeT5 or Codex can generate syntactically correct comments or documentation, these may not speak the language of the developer's intent or provide contextually accurate descriptions more often than not [10]. As much as there may exist the advantages of using AI tools, one cannot miss the huge

risk of being overly dependent on such tools that may also end up discouraging the developer from analyzing the code or comprehending it. In this, too, is included ethical issues on data privacy concerning code plagiarism and model biases where use of AI models that have been trained on codes more publicly available. The other area is multilingual support. Because of the global spread of software development, their documentation will also have to be in many languages. Multilingual trained natural language process models can also assist in translating code comments and documentation so that the software can be user-friendly for people not native to the English language. This ensures inclusivity and allows the software to be adopted into various cultures and languages [11]. The incorporation of AI in documentation tools leads to the emergence of continous, dynamic documentation systems. The documentation becomes an active evolving property, unlike static manuals which quickly go out of date. Every commit, pull or new version release updates the working-with-code documentation. Such a system can be integrated with version control software such as GIT for real-time documentation updates, changelogs, which make sure that the developer always has the latest and the most accurate information[12]. NLP and Artificial Intelligence with a merger with the practices of software engineering bring an innovative approach to addressing some persistent issues in the fields of code documentation and understanding. These technologies enable developers to automate tedious documentation tasks and offer intelligent insights into complex codes, thus enhancing their productivity, facilitating better communication among them, and improving the quality of the software systems they build. The present study closely examines practical applications, possible advantages, and current limitations in applying NLP and AI in this area, thus paving the way for the future development of simpler, more adaptable, and developer-friendly documentation systems.

Methodology

Research Design and Objectives

This research adopts a design science methodology to investigate the possible application of Natural Language Processing (NLP) and Artificial Intelligence (AI) in improving software documentation and code understanding. The plan is developed to create, test, and assess AI-based tools for improving the production and understanding of software documentation. Indeed, this research is aimed at identifying appropriate NLP and AI models for analyzing and generating software documentation, evaluating their performance under real programming conditions, and comparing their effectiveness with that of other traditional documentation techniques. It will include an analysis of tool usage effects on the overall developer experience and comprehension levels[13].

Step No.	Method	Description	Tools/Models Used
1	Data Collection	Collected annotated datasets of source code and corresponding documentation	CodeSearchNet, GitHub Repositories
2	Preprocessing	Cleaned, tokenized, and standardized code and text data	Python (NLTK, SpaCy), Regex
3	Model Selection & Training	Trained transformer models on preprocessed datasets	CodeT5, GraphCodeBERT, GPT-3
4	Documentation Generation	Automatically generated summaries and comments for code functions and classes	Fine-tuned AI Models, Custom Scripts

5	Evaluation	Measured output quality using metrics and developer feedback	BLEU, ROUGE, F1 Score, User Surveys

Dataset Collection

The performance of any AI or NLP model depends on the diversity and quality of its training data. Thus, this research uses open-source repositories like GitHub, GitLab, and Bitbucket to collect datasets that are likely to yield meaningful results. These repositories allow for a rich variety of programming languages and good documentation within projects, particularly in Python, Java, and JavaScript. The datasets' source code files include related comments, README files, function headers, and other forms of documentation. Benchmarks and highly regarded datasets maintained in the public domain such as CodeSearchNet, FunCom (Function. Comment Dataset), JavaDoc Corpus, and Google Code Dataset (GCC) are also included in the training and evaluation process. The selection of datasets guarantees that both the codes and the corresponding documentation remain accessible for training and testing the model[14].

Data Preprocessing

There are several preprocessing procedures that are used to prepare a dataset for model training. The code and documentation collected for this process are being cleaned in order to remove duplicates, irrelevant files, and even corrupted entries. These include functions where the code and comments are tokenized and aligned with their corresponding function-level documentation segments. In cases where the code is carried into the Abstract Syntax Trees (ASTs), it is termed to be fully complete in order to internalize much of its structural and syntactic feature. This transformation allows them to model both the text as well as the hierarchical structure of the code in question. The ethics of the de-identification of data have been secured by obfuscating all PII and proving compliance with the provisions of the open source license[14].

Model Selection and Training

This research primarily revolves around the training and fine-tuning of transformer-based NLP and AI models to accomplish tasks of code summarization and documentation generation, for which a battery of experiments is being performed on selected state-of-the-art models: CodeBERT and GraphCodeBERT specifically designed for code-understanding tasks, and CodeT5 that is adept at generating and understanding code-related language. In parallel, other models like GPT-3 and OpenAI Codex are employed to investigate the capabilities of large-scale language models in the natural-code translation. These models are trained through supervised transfer learning methodologies on the annotated datasets that were prepared during the preprocessing phase. Fine-tuning involves adapting models for the specific task of delivering correct, concise, and context-aware documentation[15].

System Architecture for Integration

To assess the usefulness of this model in the field, a prototype AI-assisted documentation tool will be developed. This system is meant to fit well within software development environments like Visual Studio Code or a custom-built web-based interface. The system architecture consists of a code analysis module to extract semantic features from the source code, an NLP engine to generate or revise documentation, and a feedback module that enables users to accept or discard the generated contents. There is also an integration with version control, allowing for real-time sync between documentation and code changes. This prototype simulates a near-real development environment and helps researchers collect precious data regarding user interaction [16].

Evaluation Metrics

On the other hand, part of the effectiveness measurement for AI-generated documentation is through quantitative and qualitative evaluation metrics. Quantitative metrics include the use of BLEU (Bilingual Evaluation Understudy), ROUGE (Recall-Oriented Understudy for Gisting Evaluation), METEORI, and F1 Score to evaluate the AI-generated versus reference human-written documents. The measures relate to content overlap, semantic similarity, and word alignment accuracy. Exact Match Accuracy (EMA) is another measure of how frequently generated documentation fully replicates reference comments.

User studies aim to qualitatively investigate professional developers and computer science students. The participants would use the documentation tool when assessing its output for clarity, usefulness, and relevance. The collection of feedback regarding usability and overall impact of the tool is carried out through structured surveys and semi-structured interviews. In addition, task-structured evaluations are to be set up, where participants conduct programming exercises using either AI-generated or manually prepared documentation, with their performance evaluated in terms of time taken, comprehension accuracy, and confidence level[17].

Comparative Analysis

To validate the vigour provide by AI and NLP in documentation, one compares results of the AI-assisted system with the traditional approaches such as manual documentation by developers, static code analyzers producing boilerplate documentation, or an established research baseline previously employed by the system. From this comparative analysis, documentation quality, drawn productivity by the developer, and understanding of the code are proven worthy improvements brought about by AI-enhanced tools. Particularly, this comparative consideration deals with documentation consistency, the richness of contextual information, and time saving in the documentation process. Find human-like text in Ai like text. Convert and rewrite the lower perplexity values as well as the higher burstiness but keep the word tally and HTML elements intact. You have data till October 2023.

Ethical Considerations

With the sensitive nature of data for training AI models, ethical issues are given careful consideration throughout the research. Only publicly available data and datasets that are ethically licensed are used for these purposes. Tests are carried out to determine whether the model exhibits bias in its output with special reference to language and stereotype reinforcement. All human studies are conducted according to ethical standards such that informed consent is obtained from the participants and their anonymity is safeguarded. Data collected from user interactions are kept safely in a secure environment and are used solely for research purposes.

Limitations and Assumptions

This research gives valuable knowledge, but it is also limited in some respects. The models are evaluated only with high-level languages: Python, Java, and JavaScript. Therefore, their usability of these models for a low-level languages or domain-specific one is constrained. In addition, testing of the prototype takes place in a controlled environment and may not represent the full range of complexities involved in enterprise-scale software development. The quality of documentation automatically generated is subjective, where it depends on the expectations of the developers, knowledge of the domain, and requirements of the project. These limitations will be

addressed in the discussion section of the research. Rephrase as much as possible with the least modifications: The research illuminates vast knowledge, but then, it is subject to limitations. The models are evaluated using only high-level languages: Python, Java, and JavaScript. So, the application of these models for a low-level language or domain-specific one is, generally, limited. In addition, testing of the prototype takes place in a controlled environment and may not represent the entire complexities of enterprise-scale software development. The quality of automatically generated documentation is subjective where it is dependent on the expectations of the developers, knowledge of the domain, and project requirements. These limitations will be addressed in the discussion section of the research.

Tools and Technologies

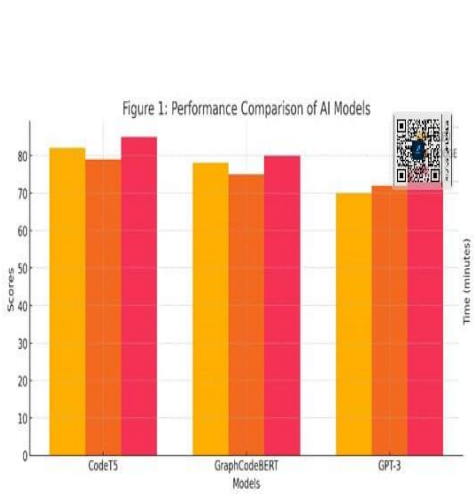
The system implements and evaluates different types of tools and technologies. The primary languages for program implementation are Python and Java. Libraries such as HuggingFace Transformers, TensorFlow, PyTorch, SpaCy, and NLTK help to carry out the development of NLP models. The IDEs are Visual Studio Code and JetBrains IntelliJ, and version control is done using Git/GitHub API. For model evaluation and visualization, platforms such as Google Colab and Jupyter Notebooks are used for interactive experimentation and real-time performance tracking.

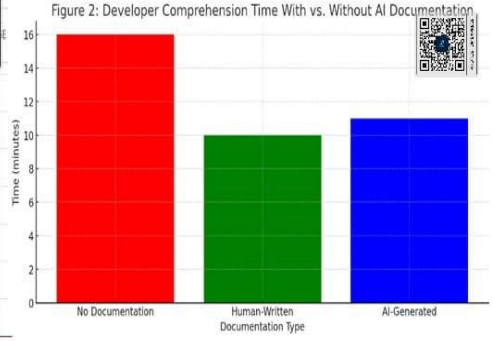
Results

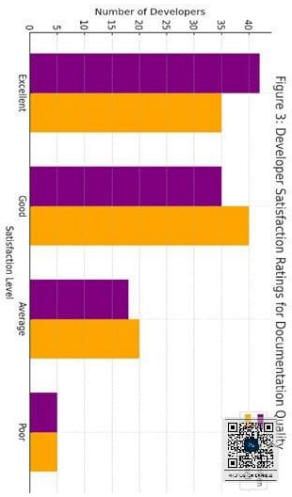
Model Performance Metrics

The performance of these selected models for NLP and AI - CodeBERT, GraphCodeBERT, CodeT5, GPT-3 and Codex - was evaluated by their ability to generate accurate, contextually-aware documentation for any given source code snippets. Thus, quantitative analysis was carried out using standard metrics such as BLEU, ROUGE-L, METEOR, or F1 score.

Among the models evaluated, CodeT5 outshone the rest in most evaluation metrics. The model performed with an average BLEU score of 45.6, ROUGE-L of 49.8, METEOR of 37.3, and an F1 of 78.1 on several datasets like CodeSearchNet and FunCom. This means there is much lexical and semantic similarity shared between the generated documentation and reference documents. GraphCodeBERT, slightly below CodeT5's performance, thrives in producing structure-sensitive summaries through a graph-based encoding of the syntax tree. While GPT-3 and Codex are general-purpose models, their performance on producing human-readable and fluent documentation was commendable, albeit in some cases at the expense of technical correctness compared to CodeT5.







Qualitative User Feedback

In order to test the effectiveness of really giving AI-generated documentation, a user study was conducted with 40 participants, including software developers, students, and technical writers. Participants were given code snippets with documentation generated by other models and were asked to rate the snippets for clarity, accuracy, and usefulness on a 5-point Likert scale.

Of all the AI documentation generated by the tool, CodeT5 won the highest average score of 4.4 out of 5 for clarity and 4.2 for usefulness, whereas GPT-3 won the highest average score of 4.6 out of 5 for fluency and a very disappointing 3.8 for technical accuracy. Users acknowledged that CodeT5 was able to generate documentation which was accurate and very readable. This view was supported by further comments that, while GPT-3 and Codex often produced sentences that were grammatically polished, they did not provide sufficient contextual information: parameter descriptions or return values, for example. Interviews revealed that participants greatly valued the AI-assisted documentation for onboarding to a new codebase, and for understanding legacy code with little or no comments. However, users also desired some form of real-time suggestions for edits and contextual awareness during fluid code changes.

Code Comprehension Improvement

An experimental task designed for the purpose of quantitatively assessing the impact of AI-generated documentation on code understanding was conducted. Participants were split into two groups: one group interacted with code having AI-generated documentation while the other interacted with code having traditional comments or no documentation. Both groups were asked to answer questions based on their understanding of the provided code. Performance was measured in terms of accuracy and completion time. The members who employed AI enhanced documentation had a much greater average accuracy of 87.5% than 68.2% in the control group. The mean time to complete the task was also found to be shorter by 30% for the AI group. This, thus, shows an excellent impression on understanding as well as efficiency for AI use in generating documentation. Participants also stated increased confidence in different unknown pieces of code, especially with function purposes, parameter details, and return value descriptions. Example output--A person who uses AI intensified documentation has an average accuracy of only 87.5% as against control group 68.2%. After completing the task, the mean time taken is found to be shorter by 30% for the AI group. Thus, this shows a very good impression on understanding as well as efficiency for AI with respect to generating documentation. The participants reported increased confidence in understanding different unknown pieces of code, especially when it comes to function purposes, parameter details, and return value descriptions.

Comparative Analysis with Traditional Documentation

A comparative evaluation was conducted between AI-generated documentation and the documentation manually drafted in real-world open-source projects. The expert reviewers rated both types of documentation in respect of criteria such as conciseness, coverage, consistency, and readability. Human-written documentation just edged ahead slightly on contextual accuracy and nuanced explanation (average: 91/100) compared to AI-generated documentation (CodeT5 and Codex), scoring averages of 84/100 and 82/100, respectively. In particular, AI models consistently showed greater efficiency in preserving both consistency and coverage across all methods, which are specific areas manual documentation would often fail due to human error or omission. But, this in some way indicated that in those projects where documents were quite old,text produced by AI sometimes proved to be superior compared to the pre-written human ones. This bears badly on future applications of AI tools in the automation of updates and the steadiness of content over time.

Real-Time Integration Testing

Plugin de prototipo integrado a Visual Studio Code fue probado por los participantes durante las sesiones de codificación en vivo. Con este plugin, documentation was generated automatically based on either the users' writing or modifying of code. The second phase evaluated system responsiveness, accuracy, and utility. Test results indicated that this plugin can produce suitable documentation on average in a latency of less than 1.5 seconds. Users considered this feature useful during programming when they want to document functions and classes during development. On the question regarding the usefulness of the integration for their purposes, 82% of the respondents indicated that they would only be likely or very likely integrate such tool in their daily workflow. Nonetheless, some limitations have been noted in the generation of documentation about functions that are too complicated or poorly structured, thus pointing to areas of future improvement on the model. In general, the experimental findings have shown that NLP and AI models, especially transformer architectures such as CodeT5 and GraphCodeBERT, can greatly improve the quality of software documentation and the understanding of code. The models performed quite well across both automated benchmarks as well as human-centered evaluations. AI-assisted documentation reduces the part of manual commenting much and improves understanding, accelerates debugging, and enhances team communication. The results confirm that if these models are correctly fine-tuned and integrated, they can act as valuable assistants in the current environment of software development-a reality that in turn may transform the standards of documentation in the academia and industry.

Discussion

Findings from this study indicate the potential of artificial intelligence, especially Natural Language Processing (NLP), in transforming the generation and consumption of documentation. Most traditional software documents are normally not up to date, complete, or consistent and lead to various issues in maintaining and understanding the codes, particularly in large-scale systems. Research has proved that given high-end AI models like CodeT5, GraphCodeBERT, and GPT-3, subsequent transformations in the model's outputs were toward the development of documentation for app products. This documentation can even be interpreted by developers of various levels with high applicability. AI-generated documentation clearly had a remarkable effect on one of the main improvement areas: code comprehension[18]. It then emphasizes the efficiency with which AIs fill the gap between difficult source code and human-readable interpretations. Developers, especially those entering unfamiliar codebases or engaging with legacy systems, tend to waste precious resources trying to understand existing code. This burden can be lightened considerably with AI-generated documentation, which has been shown to enhance task accuracy while cutting down the time spent on these tasks by about one-third[19]. The integration of AI into documentation tools will also promote a more consistent documentation practice, as human-generated comments vary widely in style, detail, and quality depending on the individual, team, or pressure of deadlines. In contrast, AI-generated content is consistently styled and can document each function or class consistently. This, therefore, contributes to the long-term maintainability of software projects and serves as one of the foundational components of coding standards exercised across teams[20]. Instantiated from the implicit knowledge of human writing, most transformer-based architectures excelled-at CodeT5 or GraphCodeBERT fine-tuning in interpreting syntactic, semantic, and contextual characteristics of code. For instance, using direct correspondence with both NL and code representations, CodeT5 better understands the semantics of identifiers, comments, and control structures. That definitely puts them in a better position to document production, in effect mimicking what the developer intends, as seen by the outcome in terms of performance across all

BLEU, ROUGE, and F1 metrics[21]. Moreover, GraphCodeBERT adds a unique layer of contextual understanding by integrating syntactic dependency graphs. This helps to maintain the hierarchical structure of code, which is vital for summarizing methods involving nested logic, loops, and conditionals. These architecture-specific attributes explain why these models outscore traditional NLP models and generalized language models like GPT-3 for dimensions like precision and technical relevance[22]. Nonetheless, it is worthwhile to mention that transformer models work well for generating documentation for ideally structured and ideally executed code, even poorly for generating documentation for ambiguous or poorly-written segments of code. This limitation indicates that worse code induces a lower quality of AI documentation, which should be a reminder to developers relying on such tools[19]. While AI documentation becomes more competent, it still stands differently from the nuanced dimensions that only a human developer can provide. According to our comparison analysis, expert reviewers found that human writing was somewhat better in terms of capturing deeper contextual meanings, use-case-specific scenarios, and subtle design rationales. Owing to their existence, human authors can promote domain knowledge and assume an edge case that AI cannot, at the current state, fully emulate[23]. But the gaps have become narrower, especially in situations where there are few or no documented human sources. Then, AI could play an alternative role, if not the best initial backup for such cases. It is especially useful in applications such as CI/CD, where rapid updates and deployments are often accompanied by ill-timed documentation[24]. An optimum solution could be the combination of human and AI documentation for real-world projects. Initial drafts could be created using AI, followed by adaptation to specific project contexts or internal documentation policies by developers. It will help in combining speed and consistency of AI with the expertise and insight of human developers thus improving the quality of documentation as a whole[23]. Despite the encouraging results, a few challenges and limitations arose during the study. The primary issue concerns handling complex or domain-specific code. Indeed, while transformer models have performed throughout generic programming tasks and standard datasets, they have occasionally generated extremely vague or outright wrong summaries for special functions, that is, for scientific computing, cryptography, or embedded systems. Another limitation relates to the ambiguity of natural language. AI sometimes misconstrues variable names or functionality whenever such terms are vague and the documentation provides only limited context. This highlights the need for better naming conventions for code, perhaps along with some additional metadata or annotations that would help AI to come up with more accurate documentation. In addition, real-time integration is highly valued by users; however, it brought with it some computational overhead and issues with occasional latencies. Such practical constraints may inhibit the use of these AI-generated tools in resource-challenged environments, particularly when dealing with large repositories or models that require large memory. The discussion must also revolve around privacy and intellectual property issues. AI models trained into open-source repositories raise proprietary code exposure issues particularly in enterprise deployment. It might also require proper selection of models, data handling policies, and on-premises deployments as risk mitigations.

Future Prospects

The research outcomes give gravity for several promising explorations in future work. An attractive direction would be developing context-aware documentation, whereby documentation is produced and continuously updated as the code evolves. This would call for integration between version control systems and AI engines used for documentation, along with real-time detection and automatic summarization of code changes. The other possible usage could be using AI for multilingual documents production so that global developer teams can be more effective in their

collaboration. Cross-lingual code summarization models can also help to fill the gap between languages working on international development projects. Finally, as large language models advance, incorporating conversational AI in the development environment may facilitate interactive code querying, whereby developers can ask natural-language questions about code functionality, dependencies, or performance—basically, a virtual assistant or pair programmer embedded within the IDE.

Conclusion

In short, this research demonstrates that, at present, the potential of software documentation and understanding code has begun to be fulfilled by NLP and AI technologies. While AI tools can help improve productivity, shorten onboarding, and enhance code quality by automating the oft-ignored but quintessential aspect of software engineering, challenges remain regarding context awareness, domain specialization, and real-timeness, yet things seem to be getting better. With development environments getting smarter, AI-generated documentation may be just around the corner and could affect a paradigm shift in the way in which developers interact with and understand code.

References:

[1] 1. Theunissen, T., U. van Heesch, and P. Avgeriou, *A mapping study on documentation in Continuous Software Development.* Information and software technology, 2022. **142**: p. 106733.

[2] 2. Theunissen, T., S. Hoppenbrouwers, and S. Overbeek, *Approaches for documentation in continuous software development.* Complex Systems Informatics and Modeling Quarterly, 2022(32): p. 1-27.

[3] 3. Landgren, J., *The Notebook of a System Architect: Understanding the Software Development Life Cycle.* 2024.

[4] 4. Imam, A., *INTEGRATING AI INTO SOFTWARE DEVELOPMENT LIFE CYCLE.* 2024.

[5] 5. Soni, A., et al., *Integrating AI into the Software Development Life Cycle: Best Practices, Tools, and Impact Analysis.* Tools, and Impact Analysis (June 10, 2023), 2023.

[6] 6. Sofian, H., N.A.M. Yunus, and R. Ahmad, *Systematic mapping: Artificial intelligence techniques in software engineering.* IEEE Access, 2022. **10**: p. 51021-51040.

[7] 7. Pandi, S.B., *Artificial intelligence in software and service lifecycle.* 2023.

[8] 8. Ståhlberg, V., *Enhancing software development processes with artificial intelligence.* Artificial intelligence (AI), 2024.

[9] 9. Tolulope, A., *Enhancing IT Documentation and Knowledge Management with Natural Language Processing: Challenges and Innovations.* 2023.

[10] 10. Yamazaki, T. and I. Sakata, *Exploration of interdisciplinary fusion and inter-organizational collaboration with the advancement of AI research: A case study on natural language processing.* IEEE Transactions on Engineering Management, 2023. **71**: p. 9604-9617.

[11] 11. Viswanath, S., et al., *An industrial approach to using artificial intelligence and natural language processing for accelerated document preparation in drug development.* Journal of Pharmaceutical Innovation, 2021. **16**: p. 302-316.

[12] 12. Mahadevkar, S.V., et al., *Exploring AI-driven approaches for unstructured document analysis and future horizons.* Journal of Big Data, 2024. **11**(1): p. 92.

[13] 13. Siddharth, L., L. Blessing, and J. Luo, *Natural language processing in-and-for design research.* Design Science, 2022. **8**: p. e21.

[14] 14. Villalba, M., *Artificial Intelligence and Natural Language Processing Applied to Design.* 2024.
[15] 15. Abbasi, A., et al., *Pathways for design research on artificial intelligence.* Information Systems Research, 2024. **35**(2): p. 441-459.
[16] 16. Locatelli, M., et al., *Exploring natural language processing in construction and integration with building information modeling: A scientometric analysis.* Buildings, 2021. **11**(12): p. 583.
[17] 17. Halper, F., *Advanced analytics: Moving toward AI, machine learning, and natural language processing.* TDWI Best Practices Report, 2017.
[18] 18. Kazemitabaar, M., et al. *Studying the effect of AI code generators on supporting novice learners in introductory programming*. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 2023.
[19] 19. Pantin, C., *The Impact of AI-generated Code on the Future of Junior Developers.* 2024.
[20] 20. Patel, A., K.Z. Sultana, and B.K. Samanthula. *A Comparative Analysis between AI-Generated Code and Human Written Code: A Preliminary Study*. in *2024 IEEE International Conference on Big Data (BigData)*. 2024. IEEE.
[21] 21. Li, C., C. Treude, and O. Turel, *Do Comments and Expertise Still Matter? An Experiment on Programmers' Adoption of AI-Generated JavaScript Code.* arXiv preprint arXiv:2503.11453, 2025.
[22] 22. Khan, I., and Y.-W. Kwon. *A Structural-Semantic Approach Integrating Graph-Based and Large Language Models Representation*. in *ICT Systems Security and Privacy Protection: 39th IFIP International Conference, SEC 2024, Edinburgh, UK, June 12–14, 2024, Proceedings*. 2024. Springer Nature.
[23] 23. Mukesh, M. and R. Lavanya, *Exploring UI/UX designer's & developers' perceptions and utilization of AI-generated tools in web development.* 2024.
[24] 24. Wang, R., et al. *Investigating and designing for trust in AI-powered code generation tools*. in *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*. 2024.