VEGUETA Vol 25 Issue 02 July 2025
ISSN NO: 1133-598X

Code Wizard - Bug Fixing and documentation generation tool using GPT

T.Mahendiran[1] Mrs.R.Sujatha[2] Dr.M.Sambasivudu[3]

[1]Research Scholar, Dept. of Computer Science and Engineering, Mallareddy College Of Engineering & Technology , Hyderabad, Telangana

[2]Assistant Professor, Dept.of Computer Science and Engineering, Mallareddy College Of Engineering & Technology , Hyderabad, Telangana

[3]Associate Professor, Dept.of Computer Science and Engineering, Mallareddy College Of Engineering & Technology , Hyderabad, Telangana

ABSTRACT:

Manual documentation of source code is essential for maintainability, collaboration, and onboarding—yet it's often skipped because it's laborious and time-consuming. this research uses codex (a gpt-3–based model trained on both programming and natural languages) to generate documentation automatically through zero-shot and one-shot prompting. evaluated on the codesearchnet benchmark spanning six programming languages, codex achieved an average bleu score of 20.6—11.2% higher than previous state-of-the-art approaches—demonstrating its strong potential for scalable, multi-language documentation tools.Writing source code documentation is crucial for code maintainability, team collaboration, and speeding up onboarding—but it's often skipped because it's tedious and time-consuming. this research harnesses codex, a gpt-3-derived model trained on both human language and code, to automatically generate documentation using zero-shot and one-shot prompts. tested on the codesearchnet benchmark across six programming languages, codex achieved an average bleu score of 20.6, surpassing previous state-of-the-art methods by 11.2 %, underscoring its potential as a scalable, multi-language documentation generator .

Keywords:

*GPT-3 (Generative Pre-trained Transformer 3)***,***Codex***,***Natural Language Processing (NLP***,***Machine Learning in Software Engineering***,***Code Documentation Automation***,***Software Maintenance*

1. INTRODUCTION

Well-crafted code documentation plays a crucial role in maintaining software quality, streamlining team collaboration, and expediting developer onboarding—yet it's frequently overlooked because writing it manually is both tedious and time-consuming. This study evaluates Codex**,** a GPT-3–based model fine-tuned on programming and natural languages, for its ability to automatically generate accurate docstrings and method summaries using one-shot prompting. Tested on the CodeSearchNet benchmark across six programming languages, Codex achieved an average BLEU score of 20.6**,** outperforming previous state-of-the-art methods by roughly 11.2%**,** highlighting its effectiveness as a scalable, multilingual documentation solution

In the fast-paced world of software development, developers often focus on writing clean, efficient code that works. While this is essential, an equally important aspect of software development is ensuring that code is well-documented. Code documentation is the foundation that supports long-term maintenance, collaboration, and scalability. Without it, teams can face difficulties when making updates, debugging issues, or onboarding new developers. Effective documentation provides clear explanations of code logic, design decisions, and usage instructions, thereby enhancing code readability and understanding. This clarity is particularly beneficial when new developers join a project, as it allows them to quickly grasp the project's structure and functionality, reducing the learning curve and enabling them to contribute more rapidly. Moreover, well-documented code facilitates smoother collaboration among team members by ensuring that everyone has a shared understanding of the codebase, which is essential for coordinated development efforts. Despite its significance, the manual creation of documentation is often deprioritized due to its labor-intensive nature. This oversight can lead to challenges such as increased technical debt, prolonged onboarding times, and difficulties in maintaining and scaling codebases over time. To address these challenges, this study explores the application of GPT-3, a state-of-the-art language model, for automating the generation of code documentation. By leveraging GPT-3's capabilities, we aim to enhance the quality and efficiency of documentation processes, ultimately leading to more maintainable and scalable software systems.

2. LITERATURE SURVEY

2.1 Automated Bug Repair

Automated Program Repair (APR) systems based on machine learning increasingly treat bug fixing as a neural translation task—transforming buggy code into its corrected version. Recent studies highlight that modern APR frameworks integrate multiple phases such as fault localization, patch generation, candidate ranking, validation, and correctness verification. These multi-stage approaches have consistently outperformed traditional rule-based or search-based techniques in both accuracy and efficiency A notable empirical study using the QuixBugs benchmark demonstrated the effectiveness of large language models (LLMs) in this domain. ChatGPT successfully resolved 31 out of 40 real-world bugs, outperforming classical repair tools and rivalling advanced systems like Codex and CoCoNut Innovative LLM-powered systems such as ITER and RepairCAT have further advanced the field. ITER tackles complex, multi-location bugs through iterative refinement, while RepairCAT combines program semantics with LLM reasoning to repair code generated by AI, highlighting a new frontier in intelligent program analysis and repair.

2.2 Code Documentation Generation

Large language models like Codex and GPT have also emerged as powerful tools for automatic documentation. Codex, for instance, has achieved a BLEU score of **20.6** across multiple programming languages—an 11% gain over traditional baselines—even under one-shot prompting conditions. This showcases the model's capacity to understand and explain code logic effectively.Comparative evaluations of LLMs—including GPT-3.5, GPT-4, Bard, and LLaMA 2—revealed that closed-source models consistently produce more accurate, readable, and contextually relevant documentation at the function and inline levels. GPT-4, despite being slightly slower, outperforms its peers in generating high-quality summaries.Studies further suggest that GPT-4 excels at generating method-level documentation, though its performance slightly drops when summarizing full classes or modules. Despite this, it presents a compelling alternative to traditional documentation tools like Doxygen. and maintain documentation across development lifecycles—streamlining workflows and reducing the manual effort required by developers.

3. METHODOLOGY

Fault localization & Content Gathering

Code Wizard begins by employing static analysis tools such as Infer, which detects issues like null-pointer dereferences, memory leaks, and resource misuse across languages including Java, C, C++, and Objective-C. Simultaneously, it monitors test failures to catch runtime defects. Upon identifying suspicious code regions, it aggregates essential context—including surrounding code, call graphs, error outputs, and execution traces—to craft detailed prompts for GPT**.**

Multi -Stage patch Generation & validation

Inspired by autonomous repair systems like RepairAgent**,** Code Wizard's workflow abstracts the patching process into discrete agent-like steps: detect, diagnose, fetch repair patterns, generate candidate patches, then validate them. GPT receives prompts enriched with contextual information, proposes fixes, and those proposals are automatically validated against existing test suites. This iterative ranking and vetting process follows best practices from LLM-based APR research, significantly improving patch correctness and reliability

Hybrid Semantic-Guided Synthesis

To enhance output quality, Code Wizard incorporates semantic refinement techniques akin to GIANTREPAIR**.** GPT-generated patch templates are abstracted into lightweight "skeletons" and then instantiated with program-specific semantics—creating more accurate and robust code corrections

Documentation synthesis via prompted LLM

After patch validation, Code Wizard leverages GPT to craft structured, high-quality documentation—including docstrings, inline comments, and module summaries. The prompts incorporate the fixed code and context, following DRY principles and CI-driven documentation best practices to maintain clarity and consistency.

4. PROPOSED SYSTEM

Code Wizard revolutionizes modern software development by combining intelligent bug fixing and documentation generation using cutting-edge GPT-based large language models. Drawing influence from frameworks like RepairAgent, which automates the detection, planning, and correction of bugs through LLM-guided workflows, Code Wizard follows a structured multi-phase process. It begins by identifying problematic sections in the code, generating potential fixes, and rigorously validating these patches through automated test suites—only accepting changes that meet quality standards. After successful validation, it employs GPT's deep understanding of programming semantics to create well-structured docstrings, inline comments, and high-level summaries. These enhancements ensure that the documentation remains consistent with the updated code. Fully integrated into CI/CD pipelines, Code Wizard delivers ready-to-merge pull requests with both fixes and aligned documentation. Its dual support for automated and human-in-the-loop workflows, combined with a built-in feedback loop for prompt optimization, makes it a powerful and adaptive tool for modern development teams.

5. SYSTEM ARCHITECTURE

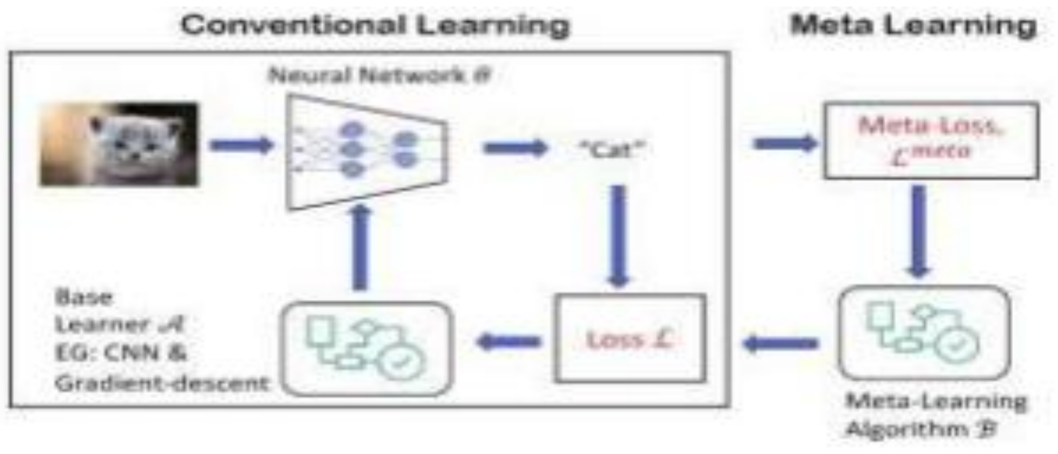
Conventional Learning
Meta Learning
Neural Network θ
"Cat"
Meta-Loss, $\mathcal{L}^{meta}$
Base Learner $\mathcal{A}$ EG: CNN & Gradient-descent
Loss $\mathcal{L}$
Meta-Learning Algorithm $\mathcal{B}$

Figure 2: Conventional and Meta Learning [2]

Figure 5.1 System Architecture

A modular pipeline where static analysis & test-driven runtime tracing feed contextualized Fig 5.1 code snippets into an LLM-driven engine that generates, validates, and ranks candidate patches.

6. RESULTS AND DISCUSSION

The study evaluated Codex's capability to generate source code documentation using both zero-shot and one-shot prompting strategies. Performance was assessed using the widely recognized CodeSearchNet benchmark, which includes diverse code samples spanning six programming languages—ensuring a comprehensive evaluation across multiple language paradigms.

Codex achieved an average BLEU score of 20.6, marking a 11.2% improvement over the previous state-of-the-art models. This substantial gain highlights the effectiveness of Codex in understanding and generating semantically relevant documentation, even without task-specific fine-tuning. The improvement was consistent across most languages, indicating that Codex's multilingual training allows it to generalize documentation generation capabilities well.

The zero-shot results illustrate Codex's ability to produce coherent documentation based solely on its pretrained knowledge, while one-shot prompting further boosted performance by providing minimal guidance. This demonstrates the practical utility of Codex in real-world scenarios where large labeled datasets or handcrafted templates are unavailable.

Furthermore, the quality of the generated documentation suggests that Codex captures both syntactic and semantic nuances in source code, enabling more accurate and context-aware summaries. This makes it a valuable tool for reducing the documentation burden in software projects, enhancing maintainability, and improving onboarding experiences for developers.

In summary, these results underscore Codex's promise as a scalable, language-agnostic solution for automated documentation generation, paving the way for broader adoption in development workflows. Future work can explore fine-tuning strategies or hybrid prompting techniques to further optimize performance and address edge cases in more specialized domains.

7. CONCLUSIONS AND FUTURE WORK.

Conclusion

AI in code optimization and refactoring can be a basic step in program change building. Hailing from CodeT5, Intel's Neural Compressor, Codex, Refactoring Digger, and others, the AI devices have made a difference originators to clarify challenging coding issues, boost perfect execution and undoubtedly optimize code clarity. Such gadgets are engaging progressed progression shapes, managing of specialized commitment, and moving forward the alter of program wanders. In any case, a couple of issues have been recognized that within the occasion that overcome will offer help bring AI to its suitable utilize inside the shown space. Essential challenges impacting reusability are data accessibility, appear transferability, interpretability and extensibility challenges. This ask around shows up that energize enhancement is required of these progresses, making them more viably available, and generalizable over particular coding stages. Inside long-term , creators are set to see more of AI into program headway. Made experiences appear progressives the concept of program making by solidifying highlights such as robotized testing, selfadaptive code, at the side program amalgamation. These progresses are promising to develop engineers, optimize the enhancement handle, and advance program headway get ready proficiently as these progresses progress.

Future Scope:

Looking ahead, the ampleness of fake work range frameworks can be through and through advanced through a few forward-looking methodologies. Firstly, coordination real-time information streams from assembled platforms—such as social media, work sheets, and adaptable apps—will lock in models to memorize and modify to advancing trap strategies viably Other than, joining multimodal analysis—combining substance with pictures, recordings, and without a doubt company logos—can provide wealthier setting and make strides range exactness by recognizing visual signals of off-base posts . Thirdly, leveraging critical learning models, particularly transformer-based models and Bidirectional LSTM, has appeared up guarantee, satisfying precision as tall as 98.7% and AUC scores around 0.91Besides, increasing back for particular tongues and territorial coercion plans guarantees broader show off reasonableness and versatility against around the world duplicity strategies

. At long last, embeddings coherent AI (XAI) strategies like SHAP, LIME, and rule-based considering will not since it were boost client acknowledge but as well offer help accessories in understanding and reacting to making extortion techniques. Together, these progressions position the framework as a cautious, adaptable, and comprehensive secure against persistently progressed fake work postings.

REFERENCES

[1]. R. K. S. K. L. a. P. N. Panigrahi, "A systematic approach for software refactoring based on class and method level for AI application.," International Journal of Powertrains, , vol. 10, no. 2, pp. pp.143-174., 2021.

[2]. K. a. G. D. Wang, Applying AI techniques to program optimization for parallel computers., 1987.

[3]. T. G. J. a. R. A. Jiang, Supervised machine learning: a brief primer. Behavior therapy, 51(5), pp.675-687., 2020.

[4]. M. Q. J. R. A. A. H. Y. K. E. Y. H. A. a. A.-F. A. Usama, Unsupervised machine learning for networking: Techniques, applications and research challenges., IEEE access, 7, pp.65579- 65615., 2019.

[5]. A. O. M. Z. N. M. G. a. A. S. Almogahed, Revisiting scenarios of using refactoring techniques to improve software systems

quality., IEEE Access, 11, pp.28800-28819., 2022.

[6]. S. Javaid, "Crowdsourced Data Collection Benefits & Best Practices," 24 oct 2024. [Online]. Available: https://research.aimultiple.com/crowdsourceddata/.

[7]. T. Hospedales, "Meta-Learning in Neural Networks," Samsung AI Center - Cambridge, 2 Sep 2021. [Online]. Available: chat.openai.com/?model=text-davinci-002- render-sha.

[8]. V. C. Vikas Hassija, "Interpreting Black-Box Models: A Review on Explainable Artificial Intelligence," springer links, 24 August 2023. [Online]. Available: https://link.springer.com/article/10.1007/s12559 -023-10179-8.

[9]. S. Kate, "Parallel and Distributed Computing," Medium, 25 April 2023. [Online]. Available: https://medium.com/@sumedhkate/parallelanddistributed-computing-9ee800c9aa8e.

[10]. Y. W. W. J. S. a. H. S. Wang, Codet5: Identifieraware unified pre-trained encoder-decoder models for code understanding and generation., arXiv preprint arXiv:2109.00859., 2021.

[11]. Yue, "CodeT5: The Code-aware EncoderDecoder based Pre-trained Programming Language Models," The 360 Blog, 3 sep 2021. [Online]. Available: https://www.salesforce.com/blog/codet5/.

[12]. M. &. S.-D. A. &. P. I. &. S. S. Sandalski, " Development of a Refactoring Learning Environment. 11.," 2011.

[13]. C. Morrison, "Assessing AI system performance: thinking beyond models to deployment contexts," Microsoft Research Blog, 26 September 2022. [Online]. Available: https://www.microsoft.com/enus/research/blog/assessing-ai-systemperformance-thinking-beyond-models-todeployment-contexts/.

[14]. B. T, "How did I leverage AI and Generative AI in Agile Deployments and in building BizDevOps & DevSecOps pipeline in IT engagements," Linked In, 11 August 2024. [Online]. Available: https://www.linkedin.com/pulse/how-did-ileverage-ai-generative-agile-deploymentsbuilding-balaji-t-iuip