AI-driven Software Design

Hou Yifei

April 13, 2024

JC3510 - Intelligent Software Implementation
Assessment: A1. Project Report

Student Name: Hou Yifei
ID: 50079704

University of Aberdeen

1 Introduction

With the advancement of AI technology, the field of software design is undergoing revolutionary changes. AI's impact is twofold: in areas such as coding practices, testing, and documentation, it is rapidly developing and becoming increasingly practical and mature. Meanwhile, in domains like project management, quality assurance, deployment, maintenance, requirements gathering, system design, security, and user training and support, AI still represents the frontier of current research and exploration.

Leveraging cutting-edge AI technology, AI-driven software design can improve the efficiency and effectiveness of automated tasks in the design phase. By accumulating vast amounts of software design knowledge, AI systems can identify potential problems, optimize design choices, and recommend best practices. This not only makes the development process more efficient, but also improves the overall quality of the software by preventing potential problems. Therefore, AI has become an indispensable tool and consultant in software design and development, indicating the future development direction of software innovation.

This article will cover the key technologies in AI-driven software design, AI performance across design stages, and introduce the AutoDev platform.

2 Key Technologies in AI-driven Software Design

Natural language processing (NLP), large language models (LLMs), and benchmark datasets form the three cornerstones of AI-driven software design. NLP enables computers to comprehend human language, LLMs like OpenAI's GPT-4[1][2] facilitate automatic programming with superior efficiency and accuracy, and well-known datasets[3] are crucial for training and validating these AI models.

2.1 Natural language processing

In the late 2010s, the development and application of BERT and GPT greatly promoted the development of the field of natural language processing (NLP). BERT is good at understanding text context, and GPT is better at generating coherent text.[4][5]

The achievements of NLP models like BERT and GPT have spurred the use of pre-trained models for programming languages.

CodeBERT is trained on GitHub data, covering six programming languages and includes code and its documentation. This means that CodeBERT learns from code examples that are explained or described in text written by humans. By training on these code and text pairs, CodeBERT can better perform tasks such as searching code using natural language queries or generating descriptions of code snippets. This approach helps CodeBERT understand how programming languages and natural languages relate to each other, which is useful for developers and others interested in translating between these two types of languages.[6]

IntelliCode Compose is a code generation tool powered by a variant of the GPT-2 model called GPT-C. GPT-C, trained on more than 1.2 billion lines of source code on GitHub, enables IntelliCode Compose to understand and generate syntactically correct sequences of code markup. The technology facilitates features like code auto-completion and generation of entire lines of code, which greatly aids programming tasks. GPT-C captures the subtle differences in coding patterns between different programming languages, improving developer productivity by predicting and completing code based on user-provided context.[7]

2.2 Large language model

Large language models (LLMs) have advanced in automatically generating code and text due to four innovations: word embeddings that enhance language understanding by grouping similar words, the Transformer architecture which allows consideration of entire text contexts, access to vast internet text for extensive learning, and pretrained models that learn language patterns and are fine-tuned for specific tasks.[8]

The paper [9] examines large-scale language models on LeetCode and TheoremQA benchmarks, showing GPT-4's superior performance over other models (Figure 1)[9]. Fine-tuned models also perform well, showcasing AI's advancements and the impact of fine-tuning on complex reasoning tasks.

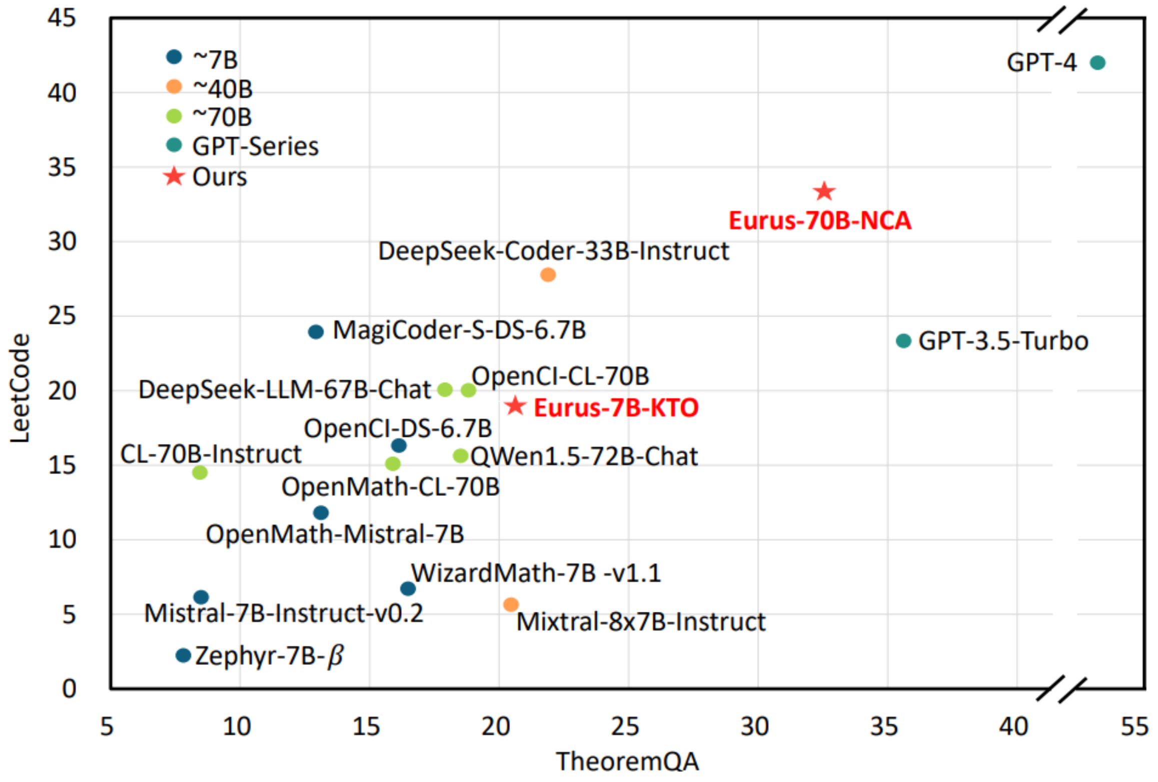

Figure 1: Comparison of large language models on LeetCode and TheoremQA benchmarks.

2.3 Benchmark datasets

Benchmark datasets are the basis for evaluating and benchmarking various algorithms and models.

In the field of AI-driven software design, various well-known benchmark datasets such as Devign[4], CT-max/min[6], Bugs2Fix[10] and CodeSearchNet[11] have played an important role. These datasets facilitate research aimed at improving the quality and productivity of software engineering. The most important functions of a dataset are:

- **Training role:** Provide extensive data for AI model training, enhancing coding and automation skills.
- **Evaluation Role:** Offer a standardized framework to evaluate AI models on metrics like accuracy and precision, enabling task comparisons such as code completion and error detection.

CodeXGLUE stands out as a unique benchmark dataset that offers a wide range of programming tasks, from code search to repair, across multiple languages. As a platform, CodeXGLUE evaluates model performance and provides baseline systems for various tasks, advancing code understanding and generation, and serving as a comprehensive resource for AI-driven software design research.[3]

3 AI's Transformative Impact on Software Design Process Stages

AI accelerates software development from concept to completion and significantly improves product quality. Advances in LLMs have elevated AI beyond a simple advisory tool, offering deep insights that help developers make informed decisions.[12]

Every stage of the software development cycle benefits from AI. Notably, areas such as automatic code generation and error detection have achieved significant gains through AI, improving efficiency and software quality. On the other hand, although the application of AI in requirements analysis and project management is still in the exploratory stage, its emerging potential has a profound impact on future development methods.

I will explore the impact of AI in software development by maturity, starting with established applications in efficiency and quality, and then highlighting its expanding roles in areas like requirements analysis and project management.

3.1 Mature Applications of AI in Software Development Processes

AI excels in automatic code generation, code documentation generation, bug fixing, test case creation, and workspace understanding. These areas demonstrate AIs ability to solve complex problems and effectively simplify the software design process through advanced LLMs.[13]

3.1.1 Automatic Code Generation

AI models, especially large language models (LLMs) such as OpenAI's GPT family and Code Llama, can generate code snippets from natural language descriptions. These models have been trained on a large number of open source code repositories, allowing them to understand programming intent in natural language and translate it into corresponding code. Programmers can be freed from tedious repetitive work, and their efficiency is greatly improved.[13]

3.1.2 Documentation Generation

By analyzing code and its structure, AI can automatically generate descriptive comments that explain what the code does. It can also create documentation for entire sections of code, such as describing the purpose of each function, the meaning of its parameters, and what its return values represent.[13]

3.1.3 Bug-Fixing

AI models can identify and correct warnings and errors flagged by static analysis tools. By examining error messages and related code context, AI proposes fixes or generates corrected code snippets, thereby improving the overall quality and reliability of the code base.[13]

3.1.4 Test Case Generation

AI can automatically generate code test cases, enhancing software quality assurance practices. By analyzing the logic within the code, AI identifies potential test scenarios and synthesizes the corresponding test scripts. This process helps ensure that the software functions as expected under various conditions. Additionally, this automation reduces the manual effort required in test case creation, allowing developers to focus more on other critical aspects of software development.[12][13]

3.1.5 Workspace Understanding

AI boosts developer productivity through two key capabilities: first, by analyzing code and documentation to gather project insights, and second, by understanding and responding to queries in natural language. This combination allows for efficient communication, enabling developers to swiftly access relevant information and collaborate effectively within their development environment.

3.2 Exploratory Use of AI Across Development Phases

A number of papers highlight applications of AI in user requirements analysis and project management. However, in other areas of software design, research remains scattered and largely theoretical. The impact of AI in these fields is expected to become more evident in the coming years.

3.2.1 Requirements Analysis

The integration of AI not only enhances the efficiency of collecting and analyzing these requirements but also broadens the scope of possibilities.

Given the diversity in user demographics such as gender, age, wealth, language skills, and health status, developers often find it challenging to fully understand the core needs of end users. Moreover, the volume of user feedback might overwhelm the development team's capacity, leading to delayed responses. AH-CID helps bridge this gap by automatically analyzing user comments to capture their true intent and relaying this information back to the developers.[14]

Furthermore, innovative applications of AI stimulate a deeper exploration of user needs. Supermind Ideator leverages AI to encourage users to explore various problem-solving possibilities, thereby enhancing their understanding and meeting their needs effectively. This tool showcases the potential of merging human designers' expertise with AI technology to refine the process of needs analysis.[15]

3.2.2 Projectment Management

AI can help project managers and teams by automating routine tasks, facilitating project analytics for better estimation and risk prediction. It offers recommendations and can make decisions, potentially transforming project management by boosting productivity and increasing success rates.[12]

A proposed framework[16] uses AI to enhance agile project management by employing deep learning models trained on historical data such as progress logs, cost estimates, and project outcomes. This approach helps managers accurately predict resources and timelines, thus reducing the risks of delays and budget overruns.

Additionally, AI enables project managers to monitor project progress in real time and adjust strategies and resources more flexibly. This approach addresses common agile management challenges like managing the product backlog, adapting quickly to project changes, and optimizing resource distribution.[16]

4 AutoDev: An Example of AI-Driven Software Design

In March 2024, Microsoft launched AutoDev, a framework for autonomous software engineering in a secure environment, showcasing recent advances in AI-driven software design.[17]

AutoDev is a versatile framework that runs in a secure environment, enabling AI agents to perform tasks such as code editing, testing, and integration to advance software development. It supports a variety of LLMs and its infrastructure can accommodate any model size and architecture, facilitating the collaboration of different AI models on specific tasks.

In the AutoDev framework, the developer's role has shifted from performing manual operations and validating AI recommendations to overseeing the collaboration of multiple AI agents and providing feedback to optimize the process. Developers set work goals by talking to intelligent robots, and AI agents complete these tasks autonomously. Developers can also monitor interactions between the AI agent and the code base through ongoing conversations to track progress against AutoDev goals. Work efficiency and quality have been greatly improved.

5 Critical Analysis

AI-driven software design is advancing rapidly, especially with the introduction of large language models accelerating its development. These models make understanding human language a core aspect of AI-driven software design.

AI technology is increasingly influential in language-related areas such as code generation, code completion, and test case creation. It is also making significant inroads into project management and user requirements analysis as part of AI-driven software design. Other stages of the software development process are poised for breakthroughs.

Prior to 2023, traditional natural language processing techniques were used to train models specifically for different programming languages, which helped with tasks such as code generation and code completion. However, the emergence of LLMs has weakened the role of traditional NLP.

LLMs face challenges such as difficulty in interpreting their decisions and ensuring the accuracy of their outputs. These issues highlight the risks of relying on them for software design, as they cannot guarantee the quality of the code or the accuracy of the AI operations. If AI-driven software design is to be used in critical areas, higher industry standards must be set.

The manual tasks traditionally performed by developers are now executed by AI agents guided by these models, introducing potential security vulnerabilities. Enhancing security could involve restricting command execution methods.

Whether generating code, creating test cases, or using artificial intelligence to manage projects, large amounts of training data from original code and project documentation are required, which inevitably raises serious privacy and security concerns.

Currently, AI-driven software design is still in the haphazard development stage, lacking an effective framework to ensure the accuracy and safety of the project. The industry needs to establish a common code of conduct to guide the development of this field.

6 Future Directions

Based on the analysis, here are potential research directions that could significantly impact AI-driven software design:

- **Development of Industry Standards for Security and Privacy:** Aim to establish comprehensive standards that blend technical, legal, and ethical aspects to foster a robust framework for AI in software design.
- **Optimization of Large Language Models:** Focus on increasing the interpretability and reliability of large language models, particularly for code generation applications.
- **Balancing Security and Functionality in AI Agents:** Investigate strategies to enhance the security and functionality of AI agents, ensuring that system integrity is not sacrificed for efficiency.
- **Improvement of Development Platforms and Environments:** Enhance platforms to better support human-AI collaboration by designing environments that improve AI's understanding and processing of human inputs, alongside maintaining high safety standards.
- **Extend AI to the entire phase of software design:** Expand AI integration into broader aspects of software design, including deployment, maintenance, feedback evaluation, project management, quality assurance, security, and user training and support.

The impacts of AI-driven software design on methodology, roles, and outcomes include:

- **Transformation of Professional Roles:** AI will take on many tasks currently handled by software practitioners, significantly altering traditional roles. Skills in communication, learning, and understanding human emotions will become increasingly critical.
- **Enhanced Software Quality and Efficiency:** Deep integration of AI into software design processes will greatly improve the quality and efficiency of software solutions, accelerate development cycles, and produce more robust, user-focused products.

- **Innovation Across Sectors:** AI-driven advancements in software design are set to foster innovation, stimulate growth, and introduce new capabilities in industries dependent on software technology.

7 Conclusion

Looking back on this research journey, my understanding of AI-driven software design has transformed from initial ignorance to a solid grasp of the concepts. At first, it was extremely challenging to determine the starting point for the research in the face of huge literature resources. However, through efficient review of literature abstracts and in-depth exploration of cited literature, I gradually opened the door to a deeper understanding of the field.

I gained a deep understanding of how AI can improve the software design process by leveraging historical code, documentation, user feedback, and other relevant data to train models. These models have greatly promoted the progress of software design by accurately identifying user intentions, guiding development behaviors, and improving execution efficiency. These insights further highlight the industrys need for new frameworks and standards to ensure that the integration of AI in software design is both sustainable and efficient.

Looking to the future, I strongly feel the potential of using AI to improve programming capabilities, which motivates me to be willing to apply the knowledge I have learned in order to achieve greater efficiency and innovation in future programming practices.

References

[1] C. Qin, A. Zhang, Z. Zhang, J. Chen, M. Yasunaga, and D. Yang, “Is chatgpt a general-purpose natural language processing task solver?” 2023.

[2] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, ukasz Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, ukasz Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O’Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. de Avila Belbute Peres, M. Petrov, H. P. de Oliveira Pinto, Michael, Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph, “Gpt-4 technical report,” 2024.

[3] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tufano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. Liu, “Codexglue: A machine learning benchmark dataset for code understanding and generation,” 2021.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” 2019.

[5] I. Solaiman, M. Brundage, J. Clark, A. Askell, A. Herbert-Voss, J. Wu, A. Radford, G. Krueger, J. W. Kim, S. Kreps, M. McCain, A. Newhouse, J. Blazakis, K. McGuffie, and J. Wang, “Release strategies and the social impacts of language models,” 2019.

[6] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, “Codebert: A pre-trained model for programming and natural languages,” 2020.

[7] A. Svyatkovskiy, S. K. Deng, S. Fu, and N. Sundaresan, “Intellicode compose: Code generation using transformer,” 2020.

[8] A. Sarkar, A. D. Gordon, C. Negreanu, C. Poelitz, S. S. Ragavan, and B. Zorn, “What is it like to program with artificial intelligence?” 2022.

[9] L. Yuan, G. Cui, H. Wang, N. Ding, X. Wang, J. Deng, B. Shan, H. Chen, R. Xie, Y. Lin, Z. Liu, B. Zhou, H. Peng, Z. Liu, and M. Sun, “Advancing llm reasoning generalists with preference trees,” 2024.

[10] M. Tufano, C. Watson, G. Bavota, M. D. Penta, M. White, and D. Poshyvanyk, “An empirical study on learning bug-fixing patches in the wild via neural machine translation,” 2019.

[11] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, “Codesearchnet challenge: Evaluating the state of semantic code search,” 2020.

[12] T. Crawford, S. Duong, R. Fueston, A. Lawani, S. Owoade, A. Uzoka, R. M. Parizi, and A. Yazdinejad, “Ai in software engineering: A survey on project management applications,” 2023.

[13] A. Agarwal, A. Chan, S. Chandel, J. Jang, S. Miller, R. Z. Moghaddam, Y. Mohylevskyy, N. Sundaresan, and M. Tufano, “Copilot evaluation harness: Evaluating llm-guided software programming,” 2024.

[14] C. Mathews., K. Ye., J. Grozdanovski., M. Marinelli., K. Zhong., H. Khalajzadeh., H. Obie., and J. Grundy., “Ah-cid: A tool to automatically detect human-centric issues in app reviews,” in *Proceedings of the 16th International Conference on Software Technologies - ICSOFT*, INSTICC. SciTePress, 2021, pp. 386–397.

[15] S. R. Rick, G. Giacomelli, H. Wen, R. J. Laubacher, N. Taubenslag, J. L. Heyman, M. S. Knicker, Y. Jeddi, H. Maier, S. Dwyer, P. Ragupathy, and T. W. Malone, “Supermind ideator: Exploring generative ai to support creative problem-solving,” 2023.

[16] H. K. Dam, T. Tran, J. Grundy, A. Ghose, and Y. Kamei, “Towards effective ai-powered agile project management,” in *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, 2019, pp. 41–44.

[17] M. Tufano, A. Agarwal, J. Jang, R. Z. Moghaddam, and N. Sundaresan, “Autodev: Automated ai-driven development,” 2024.