Proceedings of the 2nd International Conference on Software Engineering and Machine Learning
DOI: 10.54254/2755-2721/73/20240408

A systematic study in intelligent software engineering based on Large Language Model

Xiang Li

Hubei University of Technology Engineering and Technology College, No.28, Nanli Road, Hongshan District, Wuhan, Wuhan City, Hubei Province,430068, China

2302448949@qq.com

Abstract. The application and research of Large Language Model (LLM) in intelligent software engineering have become a hot topic, offering strong theoretical significance and practical value. Researchers in this field have covered various stages of software engineering. However, there is still a lack of comprehensive and holistic research analysis that unifies the current research findings, which hinders the technological development of this field. Addressing this issue, this paper conducts a comprehensive survey of LLM research in the domain of software engineering. We summarize and extract five specific subfields, including LLM and prompts, code analysis, human-computer interaction, fault prediction, and other software engineering aspects. For these representative research directions, it examines significant research works, traces their research context, and discusses their development trends. This work provides a research foundation and contextual support for subsequent software engineering development based on large models.

Keywords: software engineering, large language model, Generative AI

1. Introduction

Large Language Model (LLM) and ChatGPT, utilizing natural language processing techniques, are being increasingly employed in various software engineering tasks such as automation, natural language generation, context understanding, and code generation. These technologies assist programmers in enhancing productivity and generating more readable code and documentation.

As the utilization of LLM in software engineering is still in its nascent stages, tasks are largely unconstrained, lacking clear and organized research and summaries. There is a need to assist researchers in comprehending and mastering the developmental context of this field.

To address this gap, this paper conducts an empirical study on the application of Large Language Model in software engineering, encompassing five distinct aspects: Large Language Model and prompts, the impact of Large Language Model on code comprehension and generation, human-computer interaction, fault resolution, and software engineering in relation to LLM. This study involves several steps. First, it collects recent research and literature from these five aspects in recent years. It categorizes them based on their themes and content, and then distills and summarizes the key findings. Particular attention is given to the features of Large Language Model in the context of prompt engineering, including a broad usage survey and their influence on generation quality. However, during this research, it was also found that the quality of code generated by prompt-engineered Large Language Model varies,

© 2024 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/73/20240408

and issues such as data bias and data security persist. While prompt engineering aids large models in deep code comprehension and generation, improving efficiency and accuracy in code completion, code generation, and comment generation, it remains inconsistent in quality, lacks contextual understanding, may produce redundant content, and requires manual intervention by programmers. It also depends on a stable network environment. Furthermore, in other software generation domains such as human-computer interaction, fault prediction, and code repair, prompt-engineered large models continue to play a role. For example, some researchers have utilized the SOCRATES framework or few-shot training to enhance the intelligence of LLM, enabling them to autonomously generate interactive processes; they have used FL technology for test case generation, combined with the LIBRO framework for bug fixes; and they have significantly improved LLM function performance using the AUtoFL fault location technology. In the future, LLM will play a role in various stages of software engineering, including requirements, architecture, development, testing, and operations, relying on more comprehensive software engineering-related prompt engineering for assistance.

2. LLM and ChatGPT

Large Language Model are text generation models related to deep learning technology. There is a close relationship between them, as Large Language Model use deep neural networks to learn and simulate statistical patterns and semantic relationships in language, generating understandable and coherent text. These models are trained using extensive and massive datasets to enhance language generation, language comprehension, and language processing capabilities. Specifically, Large Language Model often employ deep learning model architectures such as recurrent neural networks like LSTM, GRU, or the Transformer. These models can handle long-range dependencies in text sequences and capture contextual information, while also extracting semantics and patterns from language through training. The training of Large Language Model involves unsupervised learning, optimizing model parameters using maximum likelihood estimation or other relevant objective functions to make the generated text conform as closely as possible to the statistical distribution of natural language. The working principle of ChatGPT involves user input as context, embedding the user's input into vector form to enable the model to understand, apply, and process it. ChatGPT employs the Transformer architecture, serving as the neural network model for ChatGPT, composed of an encoder and decoder. The self-attention mechanism in the Transformer allows the model to dynamically adjust attention weights during word generation. The model iterates in the decoder, generating coherent responses or text step by step. ChatGPT also utilizes probability distribution and sampling strategies; when generating words, it forms a probability distribution indicating the next likely word and the probability associated with that word.

3. Empirical Study

3.1. Taxonomy

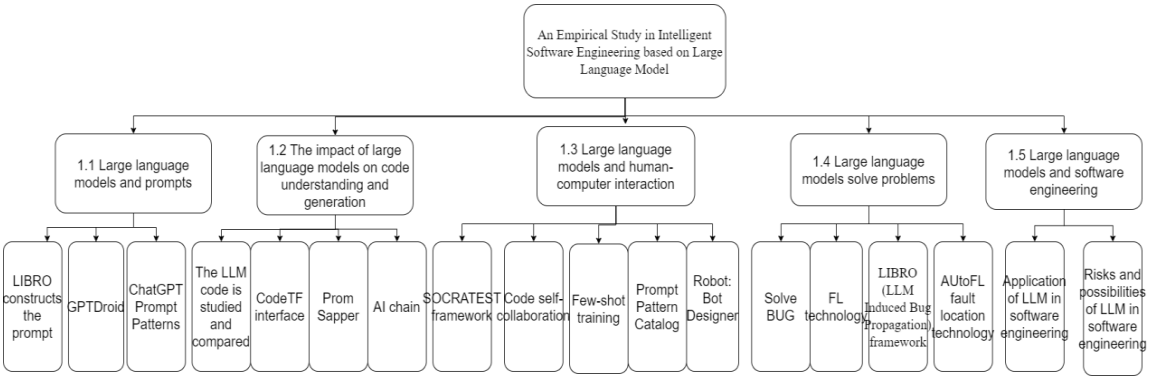


Figure 1. Large Language Model Organizational Chart.

DOI: 10.54254/2755-2721/73/20240408

As depicted in Figure 1, in the empirical research of intelligent software engineering based on Large Language Model (LLM), this paper is divided into five modules. First is the section on Large Language Model and prompts, which explores the role of prompts in Large Language Model and software engineering activities. The second module focuses on the impact of Large Language Model on code generation and understanding. Interfaces like CodeTF act as bridges between AI generation and software engineering, and frameworks like CCTEST discuss their effects on LLM. The third module covers human-computer interaction, addressing topics such as code self-collaboration and intelligent interface interaction. Frameworks like SOCRATEST and few-shot training are introduced. LLM cannot operate entirely independently from human control and currently cannot handle overly complex module programming. The fourth module delves into fault prediction and repair, with an emphasis on using LLM to locate and repair software bugs. This is achieved through techniques like FL technology and the LIBRO framework. Finally, this paper discusses the relevance of software engineering and LLM, as well as their future development and impact. The following sections provide a detailed discussion.

3.2. Application and Analysis Based on Prompt Engineering

Kang et al. constructed prompts for LLM using LIBRO from a given bug report [1]. They performed multiple LLM queries using these prompts to generate an initial set of test candidates. LIBRO was employed for test processing, allowing these candidates to be executed in the target program. LIBRO can identify and manage potential duplicate bug tests. Liu et al. discussed the role of LLM as testers, enabling LLM to interact with applications under testing [2]. They used GPTDroid, which, in each iteration of testing, first extracted the static and dynamic context of the current GUI page. They encoded LLM prompts and decoded LLM responses into executable scripts to run the app. White et al. introduced a technique called ChatGPT Prompt Patterns, a series of prompt patterns that can be used for code improvement, refactoring, requirement acquisition, and software design [3]. This technique can be applied to automation in common software engineering activities using Large Language Model (LLM) such as ChatGPT. The prompt patterns include API generators, visual generators, and more. The paper also provides a classified catalog for easy reference and application based on the type of problem to be solved.

3.3. Code Comprehension and Generation

Yuan et al. evaluated 10 open-source structural LLM for defect detection, clone detection, assertion generation, and code summarization [4]. Du et al. conducted a study on class-level code generation with 11 state-of-the-art LLM models [5]. In comparison with method-level, all LLM performed significantly better on class-level code. Li et al. used the CCTEST framework to test code completion systems and code repair in a black-box environment [6]. They designed a set of mutation strategies, PSC mutation, for generating code fragments with similar or identical program structure mutations. Bui et al. introduced the CodeTF interface, an open-source library based on the Transformer, which allows development across different models, datasets, and tasks, providing a unified interface for AI generation and software engineering [7]. Huang et al. developed AI local services using Prompt Sapper, based on large language model software engineering infrastructure [8]. This eliminates the need for an intermediary programming language and supports AI local services. Cheng et al. introduced the concept of AI chains and four activities for improving modularity and composability through the Prom Sapper approach [9]. This includes design and block views, collaborating with LLM to generate code frameworks. Practicality was demonstrated in user studies, showing a significant improvement in time efficiency. Ouyang et al. discussed the issue of non-determinism in code generation with ChatGPT [10]. They verified code generation on benchmark test sets and concluded that code candidates generated from the same instruction exhibit significant differences in correctness, syntax, and structure.

DOI: 10.54254/2755-2721/73/20240408

3.4. Self-Collaboration Framework for Code

Feldt et al. developed SOCRATEST, a session testing agent framework that operates autonomously, enabling LLM to work independently [11]. Dong et al. introduced a code self-collaboration framework where LLM take on different roles, each completing specific sub-tasks within complex operations, forming a virtual team that operates without human intervention [12]. Ahmed et al. utilized few-shot training, which has a unique collaborative role in the field of software engineering, enabling developers to quickly read code [13]. LLM can summarize specific code with minimal sample training. Yoo et al. discussed how LLM generate software specifications [14]. This paper first used Few-Shot Learning technology to induce LLM based on a limited number of examples, starting with a basic prompt construction method called FSL, which helps LLM learn how to generate standardized software text. In the second step, it explores the potential of LLM with FSL for extracting patterns in specifications. Ross et al. discussed the conversational programming assistant of Large Language Model, used by developers in software development [15]. During user interaction, the model can generate responses based on the history of the conversation. White et al. introduced the Prompt Pattern Catalog approach, which enhances ChatGPT's prompt engineering capabilities [16]. Prompt engineering is an effective skill set required for Large Language Model, forcing them to adhere to rules and automated processes, ensuring the quality and quantity of generated input. Zamfirescu-Pereira et al. created a codeless LLM chatbot called BotDesigner, which interacts with participants, guiding them through activities [17].

3.5. Fault Prediction and Repair

Kang et al. used FL technology to help LLM locate software bugs within individual failing tests [18]. The results suggest that LLM's FL and point have promising prospects and research directions in fault localization. Wang et al. discussed the use of LLM in common test cases, with a focus on test case preparation and program repair [19]. Liu et al. discussed using Large Language Model to generate test cases to address the general issue of report-to-test in software engineering [20]. They proposed the LIBRO (LLM-Induced Bug Breeding) framework, which uses OpenAI LLM Codex to generate highly reliable test cases. Tian et al. discussed the potential and feasibility of ChatGPT as a programming assistant [21]. The authors studied ChatGPT's functionality in tasks such as program repair, code summarization, and code generation. ChatGPT is effective in handling common programming tasks, but the experiments also revealed that ChatGPT has limited attention span, and its performance is poorer in handling problem descriptions. Additionally, the research found that ChatGPT can infer the original intent of the code. The experiments provide a better understanding of ChatGPT's practical applications in software engineering. Kang et al. introduced the AUtoFL fault localization technology, which significantly improves the performance of LLM functions by making appropriate function calls based on available information and generating corresponding reasons and explanations using failed test cases [22].

3.6. Other LLM-Based Software Engineering

Russo et al. discussed the complexity of applying artificial intelligence in software engineering [23]. LLM open up new possibilities for software development. Developers can leverage AI for inspection, labeling, and various data sources in programming, testing, and maintenance. There is potential value for LLM across different phases of software engineering. Hou et al. explored the applications of LLM in software engineering, focusing on aspects such as semantic ambiguity, requirement categorization, terminology identification, and other relevant areas [24]. The paper particularly emphasized the significant role of datasets in LLM-based solutions for software engineering challenges. Ozkaya et al. introduced the extensive opportunities for Large Language Model (LLM) in software engineering [25]. However, the paper also discussed the associated risks and impacts, as well as considerations for new development environments.

DOI: 10.54254/2755-2721/73/20240408

3.7. Discussion

When employing Large Language Model (LLM), prompts serve as guides and are essential for generating appropriate outputs, ensuring accuracy, relevance, and fluency. They help LLM better understand and reproduce bugs accurately. Regarding code comprehension, LLM undergo extensive learning during the pre-training phase, involving vast amounts of code and text, which imparts them with a certain degree of source code understanding. In code generation, LLM can generate code snippets based on pre-training and contextual understanding, making use of patterns and structures from the source code to produce code that meets specific requirements. In the realm of human-computer interaction, LLM is widely used for constructing powerful conversational systems. They can act as virtual assistants, providing users with responses that substantially reduce the need for manual intervention, enabling people to access information more quickly and conveniently. LLM play a valuable role in software fault resolution by providing bug localization and diagnosis. By analyzing code, error reports, and log information, they aid developers in pinpointing the root causes of faults. They also offer suggestions for error fixes, serving as references to enhance repair efficiency. However, it's important to note that LLM-generated suggestions may not always be entirely accurate or universally applicable, necessitating validation and testing. In addressing various other software engineering challenges, LLM plays roles in different phases. They can reduce the workload of developers through automated code generation, enhance code quality and maintainability through error detection and code review, and support teams in better record-keeping and requirement analysis, ultimately reducing the risk of project failures.

4. Conclusion

The intelligence of LLM in code analysis is gradually reshaping the work habits of programmers and improving software development efficiency. Various code generation approaches based on these models aim to maximize their development support capabilities. Through our research on LLM in the field of software analysis, we find that LLM exhibits both advancements and limitations. In most scenarios, they can assist developers and researchers by providing relatively accurate code generation and framework development services. However, some methods have limitations, particularly in human-computer interaction. ChatGPT, for instance, may not always deliver satisfactory results, with issues such as repetitive feedback, redundancy, and inaccurate responses. In the future, as artificial intelligence and Large Language Model technology continue to advance and scale, contextual understanding is expected to improve. Language generation and dialogue systems based on LLM will likely become more innovative and practical.

References

[1] Kang S, Yoon J, Yoo S. Large language models are few-shot testers: Exploring llm-based general bug reproduction[C]//2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). IEEE, 2023: 2312-2323.

[2] Liu Z, Chen C, Wang J, et al. Chatting with GPT-3 for zero-shot human-like mobile automated GUI testing. arXiv 2023[J]. arXiv preprint arXiv:2305.09434.

[3] White J, Hays S, Fu Q, et al. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design[J]. arXiv preprint arXiv:2303.07839, 2023.

[4] Yuan Z, Liu J, Zi Q, et al. Evaluating instruction-tuned large language models on code comprehension and generation[J]. arXiv preprint arXiv:2308.01240, 2023.

[5] Du X, Liu M, Wang K, et al. Classeval: A manually-crafted benchmark for evaluating llms on class-level code generation[J]. arXiv preprint arXiv:2308.01861, 2023.

[6] Li Z, Wang C, Liu Z, et al. Cctest: Testing and repairing code completion systems. In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)[J]. 2023.

[7] Bui N D Q, Le H, Wang Y, et al. CodeTF: One-stop Transformer Library for State-of-the-art Code LLM[J]. arXiv preprint arXiv:2306.00029, 2023.

DOI: 10.54254/2755-2721/73/20240408

[8] Xing Z, Huang Q, Cheng Y, et al. Prompt Sapper: LLM-Empowered Software Engineering Infrastructure for AI-Native Services[J]. arXiv preprint arXiv:2306.02230, 2023.
[9] Cheng Y, Chen J, Huang Q, et al. Prompt Sapper: A LLM-Empowered Production Tool for Building AI Chains[J]. arXiv preprint arXiv:2306.12028, 2023.
[10] Ouyang S, Zhang J M, Harman M, et al. LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation[J]. arXiv preprint arXiv:2308.02828, 2023.
[11] Feldt R, Kang S, Yoon J, et al. Towards Autonomous Testing Agents via Conversational Large Language Models[J]. arXiv preprint arXiv:2306.05152, 2023.
[12] Dong Y, Jiang X, Jin Z, et al. Self-collaboration Code Generation via ChatGPT[J]. arXiv preprint arXiv:2304.07590, 2023.
[13] Ahmed T, Devanbu P. Few-shot training LLMs for project-specific code-summarization[C]//Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. 2022: 1-5.
[14] Xie D, Yoo B, Jiang N, et al. Impact of Large Language Models on Generating Software Specifications[J]. arXiv preprint arXiv:2306.03324, 2023.
[15] Ross S I, Martinez F, Houde S, et al. The programmer's assistant: Conversational interaction with a large language model for software development[C]//Proceedings of the 28th International Conference on Intelligent User Interfaces. 2023: 491-514.
[16] White J, Fu Q, Hays S, et al. A prompt pattern catalog to enhance prompt engineering with chatgpt[J]. arXiv preprint arXiv:2302.11382, 2023.
[17] Zamfirescu-Pereira J D, Wong R Y, Hartmann B, et al. Why Johnny can't prompt: how non-AI experts try (and fail) to design LLM prompts[C]//Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems. 2023: 1-21.
[18] Kang S, An G, Yoo S. A Preliminary Evaluation of LLM-Based Fault Localization[J]. arXiv preprint arXiv:2308.05487, 2023.
[19] Wang J, Huang Y, Chen C, et al. Software Testing with Large Language Model: Survey, Landscape, and Vision[J]. arXiv preprint arXiv:2307.07221, 2023.
[20] Liu Z, Chen C, Wang J, et al. Fill in the blank: Context-aware automated text input generation for mobile gui testing[C]//2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). IEEE, 2023: 1355-1367.
[21] Tian H, Lu W, Li T O, et al. Is ChatGPT the Ultimate Programming Assistant--How far is it? [J]. arXiv preprint arXiv:2304.11938, 2023.
[22] Kang S, An G, Yoo S. A Preliminary Evaluation of LLM-Based Fault Localization[J]. arXiv preprint arXiv:2308.05487, 2023.
[23] Russo D. Navigating the complexity of generative ai adoption in software engineering[J]. arXiv preprint arXiv:2307.06081, 2023.
[24] Hou X, Zhao Y, Liu Y, et al. Large Language Models for Software Engineering: A Systematic Literature Review[J]. arXiv preprint arXiv:2308.10620, 2023.
[25] Ozkaya I. Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications[J]. IEEE Software, 2023, 40(3): 4-8.