A Study of Web Code Generation Based on ChatGPT

Zhan Shu[1], Zijie Dong[2]

{780315338@qq.com[1], 418541993@qq.com[2]}

School of Computer science, Hubei University of Education, Hubei, China[1]
School of Mathematics and Statistics, Bigdata Modeling and Intelligent Computing research institute, Hubei University of Education, Hubei, 430205, China[2]
Hubei Key Laboratory of Applied Mathematics (Hubei University)[2]

Abstract. With the rise of large language models (LLMs) such as ChatGPT in the field of code generation, these models have demonstrated impressive abilities in understanding code semantics and implementing complex functionalities, especially showing potential in web development scenarios. Developing web applications is a critical task widely used in interactive software systems across various fields. However, current automated web code generation still has limitations, often failing to cover complete front-end and back-end functionalities or achieve complex interactive logic. Based on this, this paper takes ChatGPT-4o as an example, constructing a comprehensive student management system to systematically analyze and evaluate its performance and applicability in generating front-end and back-end code. First, the paper outlines the system's requirements analysis and module design. Then, it thoroughly documents the entire process of generating front-end and back-end code based on ChatGPT-4o. Through this process, the paper examines ChatGPT-4o's performance in terms of code generation efficiency, functionality accuracy, and the level of human intervention required, analyzing its strengths and limitations with experimental data. The experimental results indicate that large models like ChatGPT significantly simplify code generation and accelerate the development process, yet still require human optimization when handling complex logic and interaction design. This study provides a valuable reference for web code generation based on large models and outlines future directions for improvement.

Keywords: large language model, code generation, Java, web development, software development

1 Introduction

In recent years, the rapid advancement of artificial intelligence technology has driven profound changes in the field of software development. Generative AI, exemplified by large language models (LLMs) like ChatGPT, has garnered significant attention from developers due to its remarkable performance in natural language understanding and text generation. These models not only comprehend complex natural language instructions but are also capable of generating high-quality code, facilitating the automation of the entire process from requirements to code implementation. This capability offers the potential to enhance development efficiency and reduce costs, particularly in the multi-layered, multi-module landscape of web application development, where automated code generation can significantly streamline the development workflow and accelerate software delivery. As a core form of modern interactive systems, web applications are widely used across various industries, including e-commerce, education,

CONF-MLA 2024, November 21, Adana, Turkey
Copyright © 2025 EAI
DOI 10.4108/eai.21-11-2024.2354633

healthcare, and social media. Web application development typically involves designing the front-end user interface, implementing back-end business logic, and managing and maintaining the database, incorporating a variety of technologies such as HTML, CSS, JavaScript, and Java, along with stringent requirements for logical processing and data security. However, traditional web development processes place high demands on developers' programming skills and experience. Leveraging artificial intelligence to automatically generate code that aligns with business requirements and has a well-structured architecture has become a key focus in the industry.

Although some research has attempted to use models for code generation—such as GitHub Copilot, which demonstrates relatively mature capabilities in generating code snippets—these technologies are mostly limited to generating single modules or snippets, lacking comprehensive support for an entire web application architecture, and are unable to achieve complete front-end and back-end development integration. Advanced large language models like ChatGPT, however, exhibit new potential in this area, with multi-layered generation capabilities that make it possible to automatically generate code in complex web development scenarios. Nonetheless, there has been limited work to date on using large models to develop complex, practical web applications, making it challenging to assess the current capabilities of large models in generating practical software. Against this background, this paper selects ChatGPT-4o as the subject of study and uses a student management system as an example to systematically evaluate the performance of large language models in web code generation. We begin by analyzing the requirements of the student management system, then design modular generation prompts based on this analysis, and subsequently assess the effectiveness of the generated front-end and back-end code through experimentation. Through this project, we explore ChatGPT-4o's performance in generating front-end and back-end code, implementing interaction logic, and optimizing code, as well as conducting a quantitative analysis of the usability, functional accuracy, and degree of human intervention required for the generated code. Compared with existing tools, ChatGPT not only accomplishes basic code generation but also adjusts code logic based on prompts, covering all layers from front-end UI to back-end logic. It performs well in terms of development efficiency and usability, achieving a high level of automation and accuracy. At the same time, we found that general-purpose large models like ChatGPT-4o are still insufficient in handling complex interaction logic and optimizing user experience, requiring significant human intervention to ensure smooth user interactions and responsive system performance.

Through systematic experimentation, this paper aims to provide empirical support for the application of large language models in the field of web development, shedding light on the model's applicability in practical development and its potential contributions across different scenarios. We believe that as model architectures and prompt engineering techniques continue to advance, the impact of large language models in software development will continue to grow, with the potential to become an essential assistive tool for developers and drive the evolution of intelligent development processes.

2 Literature Review

2.1 LLM-based Code Generation

In recent years, the rapid development of large language models (LLMs) has made LLM-based code generation a key research focus in software development. This technology not only demonstrates powerful capabilities in code comprehension and generation but also provides new possibilities for improving developer productivity [1,2]. LLM technologies are currently applied across diverse coding tasks, such as generating test cases for compiler testing [1], automating industrial control logic [3,4], and enhancing code generation with requirements clarification frameworks [5]. These studies have proven the potential of LLMs to significantly improve both the efficiency and quality of code generation. However, challenges remain in ensuring functional correctness, handling logic complexity, and addressing security concerns [6,7]. For example, the ClarifyGPT framework reduces the impact of ambiguous requirements by generating clarifying questions, significantly improving the functional accuracy of generated code [5]. Similarly, studies evaluating vulnerabilities in generated code, such as SQL injection or cross-site scripting, have highlighted the importance of incorporating security-focused prompt patterns [8,9]. Additionally, interactive multi-round refinement—where users iteratively adjust the initial output—has shown promise in improving the accuracy and reliability of LLM-generated code [7,10]. Despite the strong performance of LLMs in basic tasks like algorithmic code generation and single-module implementations, they require further improvement for multi-module system development and complex interactive logic handling [11,12]. Furthermore, studies have found that insufficient testing datasets can misrepresent evaluation outcomes, leading to overestimated LLM capabilities [6]. In industrial control applications, a lack of integration with proprietary function blocks often necessitates significant manual optimization of generated code [4]. To enhance overall performance, future research focuses on improving code security, generation efficiency, and evaluation frameworks. For instance, integrating retrieval-augmented methods to strengthen contextual understanding can significantly improve the accuracy and usability of generated code [4,13]. Additionally, developing user-oriented low-code platforms and multi-round interactive optimization techniques may further enhance the adaptability of LLMs in real-world development environments [14].

2.2 LLM-based Web Development

Web development is a central area of modern software engineering, encompassing multiple layers from front-end interface design to back-end logic implementation. In recent years, LLM-based automated web code generation has emerged as a promising research area, especially for lowering development barriers and improving efficiency [8,13,14]. Current studies focus on generating functional module code (e.g., back-end logic, database interaction) and detecting potential security vulnerabilities [8,9]. For example, the Kodless platform generates back-end functionality code from natural language descriptions using modular architecture principles, providing standardized application designs [13]. Additionally, advancements in prompt engineering have significantly enhanced the applicability and accuracy of LLM-generated code [14]. Research has also demonstrated that LLMs can generate complete web applications with front-end and back-end interactions [14], though challenges remain in optimizing user experience and handling complex interactive logic [9,13]. In the education sector, LLM-generated code is being explored as a teaching tool for students and instructors, aiding in

programming assignments and assessments [11,15]. While LLMs perform well in basic web development tasks, they struggle with multi-layered user interactions and complex business requirements. For example, studies highlight that LLM-generated web application code often contains security vulnerabilities, particularly in areas such as file uploads and database interactions [8,9]. Additionally, front-end code frequently requires manual refinement to meet dynamic interaction and navigation logic requirements [13,14]. To further enhance the value of LLMs in web development, future efforts should focus on several areas. First, developing more intelligent prompt templates to address complex development scenarios [5,14]. Second, leveraging multi-round interaction and contextual enhancement techniques to improve the security and functionality of generated code [9,14]. Finally, conducting user studies to evaluate the effectiveness of LLM-based development tools in low-code/no-code platforms, making web development more accessible to non-technical users [13,14].

3 Approach

3.1 Workflow

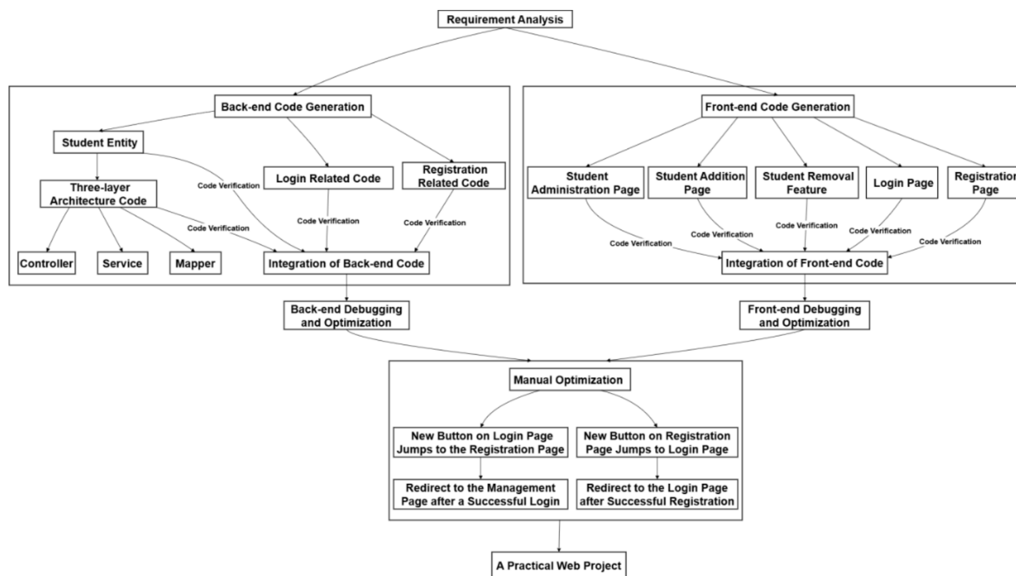

Fig. 1. Overall Workflow for Practical Website Generation Based on ChatGPT

As shown in Fig.1, the entire development process is divided into three main parts: back-end development, front-end development, and manual adjustments. The project is completed through integration, debugging, and optimization. First, during the requirements analysis phase, the project's goal is clearly defined as creating a student management system that includes features such as student management, login, and registration. This functionality is to be implemented with code generated by ChatGPT-4. Next, in the back-end code generation phase, ChatGPT generates an entity class to describe student information, a three-layer architecture code for the back-end (Controller, Service, and Mapper), and the code for handling user login

and registration requests. Each step requires verification of the generated code to ensure its correctness and functionality, ultimately forming a complete back-end system. The next stage is front-end code generation, which involves creating the user interface, including the student management page, add student page, delete functionality, login page, and registration page. Similarly, each front-end page is validated post-generation to ensure proper functionality and, finally, is integrated into a complete user interface. Although ChatGPT-4 generates most of the code, some page navigation and user interaction logic require manual adjustments to enhance user experience and system interactivity. After back-end and front-end code integration, debugging and optimization are performed to ensure logical consistency and smooth user interaction throughout the system. Through this process, ChatGPT-4 demonstrates its potential in code generation, significantly reducing repetitive work and improving development efficiency. To the best of our knowledge, we are the first to conduct the complete testing for website generation by LLM.

3.2 Prompt Templates

A prompt template is one of the key techniques in using GPT models for code generation. An effective prompt template not only ensures that the GPT model understands the user's requirements but also guides the model to generate high-quality code that meets expectations.

The design of prompt templates affects the quality of GPT-generated content. Different prompts influence the accuracy, efficiency, and applicability of the generated code. The template design must be clear and structured to ensure that the requirements for code functionality, logic, and style are effectively communicated to GPT.

3.2.1. Principles for Template Design

To create a scientifically sound and general-purpose prompt template, the following design principles should be followed:

- **Clear Functional Description:** Provide a precise description of the specific functionality of the code. For example, "Generate a user registration interface with form validation" is preferred over vague descriptions like "Generate a registration interface," as it ensures all necessary functionality is included in the generated code.
- **Technology Stack Specification:** Specify the language or technology stack to be used, such as "Generate a login page using HTML5, CSS3, and JavaScript." This helps GPT generate code aligned with project requirements.
- **Hierarchical Structuring:** Break down prompts into different levels of instructions, from high-level system architecture to specific module implementations. For instance, first generating the project's Controller, then the Service layer, and finally the Mapper layer guides GPT in generating clear three-layer architecture code.
- **Input-Output Definition:** Clearly define the input and output types for functions or modules. For example, "Generate a function for handling user login, with input as username and password, and output as login result." This helps GPT generate well-structured functions.

3.2.2 Template Examples

1) **Refining Prompt Templates:**

Detailed prompt templates not only help GPT understand requirements but also ensure that the generated code is highly applicable in specific scenarios, possesses a clear structure, and is maintainable. We break down the prompts into multi-level, detailed instructions.

2) **Functional Description:**

The functional description needs to be as clear as possible, specifying details regarding code modules, functional objectives, input, and output.

General Template:

Generate code for a [system/module] with the following requirements:

- The code implements [specific functionality]. For example: the code implements the user registration function.
- Use [technology stack/programming language]. For example: use the Java programming language.
- The input for this function is [detailed description of the input], and the output is [detailed description of the output]. For example: input is the username, password, and email address; output is whether the operation was successful and any related error messages.
- The code must have a clear structure, including [specific functionality], and possess [specific features such as scalability, performance optimization, security, etc.]. For example: the code structure must be clear and secure (check if the username exists, ensure the password meets security standards of at least 8 characters, containing both numbers and letters).

3) **Technology Stack Specification:**

It is necessary to specify the technology stack to help GPT choose an appropriate implementation approach.

General Template:

Use [specific technology stack] to generate code, with the following requirements:

- Front-end: use [HTML5/CSS3/JavaScript/React, etc.].
- Back-end: use [Java/Python/Node.js, etc.].
- Database: use [MySQL/PostgreSQL/NoSQL].
- Request format: use [JSON/XML] format to handle front-end and back-end interactions.

4) **Input and Output Requirements:**

Clearly define the input and output for each function or module to facilitate GPT in generating structured code.

General Template:

Generate a function with the following requirements:

- Input: accepts [detailed description of input type and requirements], for example: username and password.
- Output: returns [detailed description of output type and requirements], for example: operation result and status code.

- The function must [validate, process] the [specific input] and return [detailed description]. For example: the function must ensure that the username and password are not empty and verify whether the username and password match through a database query.

3.3 Code Optimization (GPT Self-Optimization + Manual Optimization)

The generated code is not always perfect, and optimization is a key step in ensuring code quality, performance, and maintainability. During the development process, code optimization can be divided into two parts: **GPT self-optimization** and **manual optimization**.

3.3.1 GPT Self-Optimization

The GPT model can not only generate code but also perform self-optimization through recursive interactions. After the initial code generation, we can use GPT again to optimize the previously generated code, typically by clearly specifying the optimization requirements.

1) Mechanism of GPT Self-Optimization

The self-optimization mechanism improves the code by repeatedly calling GPT and generating modified versions of the code. This process mainly relies on the following points:

- **Performance Optimization Prompt:** After the initial code is generated, we prompt GPT to further optimize for performance. For example, “Optimize the generated code to improve execution efficiency and reduce unnecessary calculations.”
- **Structural Optimization Prompt:** To address redundancies in the initial code, we directly prompt GPT to streamline the code. For example, “Simplify redundant code and ensure a more compact structure.”
- **Security Optimization:** We require the code generated by GPT to handle common security issues. For example, in database interaction code, prompt GPT to optimize SQL queries to avoid security issues like SQL injection.

2) Self-Optimization Example

Initial Code Generation:

Generate a login function using Java, which includes username and password verification. If the verification fails, return “Login failed.”

GPT might generate the following code:

```
public String login(String username, String password) {
    if(username.equals(“admin”) && password.equals(“1234”)) {
        return “Login Successes”;
    } else {
        return “Login Fails”;
    }
}
```

Then we prompt GPT for optimization:

Optimize the login code to add exception handling and input validation, ensuring that username and password cannot be empty and avoiding hard-coded password checks.

Optimized Code:

```
public String login(String username, String password) {
    if(username == null || username.isEmpty() || password ==
null || password.isEmpty()) {
        throw new IllegalArgumentException("The user name or
password cannot be empty");
    }
// Assume there is code here to validate the user against the
database
if(authenticateUser(username, password)) {
        return "Login Successes";
    } else {
        return "Login Fails";
    }
}
```

Through GPT self-optimization, we avoided hard coding and added input validation and exception handling, significantly improving the quality and security of the code.

3.3.2 Manual Optimization

1) **Adjustment of Page Navigation Logic**

Redirect from Login Page to Registration Page: Add a button on the login page that allows users to directly navigate to the registration page. This step ensures that users can smoothly register a new account if they are unable to log in, without needing to return to the homepage.

Redirect from Registration Page to Login Page: Add a navigation button on the registration page so that users can directly access the login page for verification after completing their registration. This design enhances the fluidity of the user experience.

Redirection After Successful Login: When a user successfully logs in, they are automatically redirected to the student management page, enabling them to manage student information directly without further manual actions.

Redirection After Successful Registration: After successful registration, users are automatically redirected to the login page, allowing them to immediately log into the system with their newly created account.

2) **Debugging and Integration**

Debugging Frontend and Backend: Conduct debugging on both frontend and backend code to ensure their interactions meet expectations. Specifically, during new user registration, ensure that the new user data is successfully written to the database and can be correctly validated by the backend. Optimization After Code Integration: After integrating the frontend and backend code, multiple tests were conducted to ensure that page redirections, data interactions, and business logic execute smoothly. For example, in the student management page, ensure that every operation—such as adding, modifying, or deleting student information—immediately reflects in the database and is updated in real-time on the page.

4 Evaluation

4.1 Experimental Details

Requirement Analysis. This work assesses the intelligent development capabilities of large models represented by ChatGPT by constructing a complete student management project. The project encompasses modules for student registration, login, and management, which include both backend data processing and frontend interaction design. Through this work, we are first aimming to comprehensively test and evaluate the performance of the ChatGPT-4o model in generating frontend and backend code, as well as its ability to handle complex logic and user interaction design. To fairly evaluate the code generation capabilities of the ChatGPT-4 model, the requirements analysis considers the following aspects:

Functionality Completeness: The student management system should include core functionalities such as registration, login, and student information management to ensure that the system covers typical web application scenarios.

Applicability: The system requirements are designed to be suitable for business management scenarios, emphasizing the practicality of the functionalities and user experience to ensure that the generated code has practical application value in real environments.

Modular Implementation: To facilitate evaluation, the requirements analysis emphasizes modular development for both the backend and frontend, such as the three-tier architecture (Controller, Service, Mapper) for the backend and the separate development of login, registration, and student management pages for the frontend. This modular division simplifies and clarifies the assessment of the generation quality for each part.

Experimental Setup. In the experiment, ChatGPT-4o was used as the large model for code generation, with Java (Java 8) chosen as the backend development language, supporting MySQL, and using MyBatis Plus for database access, along with HTML, CSS, and JavaScript for frontend development. The Spring Boot 2 framework was employed to build the backend service to achieve a standardized code structure. All generated and debugged code was executed in the same hardware and software configuration environment to ensure the fairness of the experiment.

4.2 Quantitative Metrics

Usability: Evaluate the ratio of successfully generated functional points to the total functional points to measure the ease of use of the system.

$$Usability = Fg/F \tag{1}$$

Where Fg is the number of functional points successfully generated and available by ChatGPT-4, and F is the total number of functional points required by the project.

Development Efficiency: Compare the time taken to generate code using ChatGPT-4o with the time taken for traditional manual development to assess efficiency improvements.

Human Involvement: Calculate the proportion of time spent on manual adjustments and debugging during the development process relative to the total development time.

$$HI = (Th + Tb)/Td \quad (2)$$

Where HI is the human involvement ratio, indicating the proportion of manual adjustments and debugging in the development process; Th is the time for manual adjustments; Tb is the time for debugging code; and Td is the total time required for the entire development process.

Functional Correctness: Assess the correctness and reliability of each functional module's behavior through testing.

$$FC = Fc/F \quad (3)$$

Where FC is the functional correctness, measuring the accuracy and reliability of the functional modules; Fc is the number of correct functional points passed in testing; and F is the total number of functional points.

4.3 Results Analysis

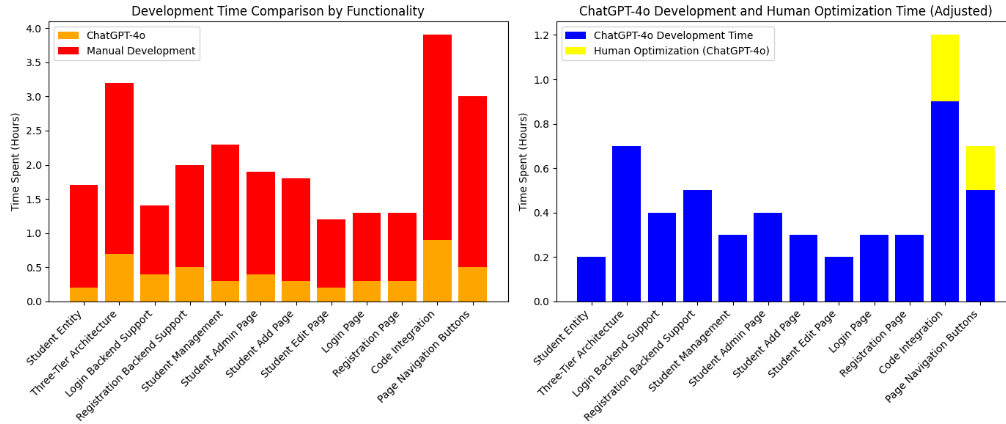

Fig. 2. Comparison of Development Time between ChatGPT-4o and Manual Development

Table 1. Analysis of Usability and Functional Correctness of System Function Points

Metrics	Function module name											
	SE	TTA	LBS	RBS	SM	SAP	SAPg	SEP	LP	RP	CI	PNB
Usability	✔	✔	✔	✔	✔	✔	✔	✔	✔	✘	✔	✔
Functional Correctness	✔	✔	✔	✘	✔	✔	✔	✔	✔	✘	✔	✔

Annotation: SE: Student Entity,TTA: Three-Tier Architecture,LBS: Login Backend Support,RBS: Registration Backend Support,SM: Student Management,SAP: Student Admin Page,SAPg: Student Add Page,SEP: Student,Edit Page,LP: Login Page,RP: Registration Page,CI: Code Integration,PNB: Page Navigation Buttons

In this work, a complete student management system was successfully generated using ChatGPT-4o, covering 12 core functional points such as login, registration, and the creation, deletion, and modification of student information. In terms of functional generation, Table 1

displays the usability and functional correctness of each functional point. Functional modules such as “Student Entity (SE),” “Three-Tier Architecture (TTA),” and “Student Management (SM)” demonstrated high usability and functional correctness. However, the functional correctness of the “Registration Page (RP)” and “Registration Backend Support (RBS)” was relatively low, particularly the “Registration Backend Support,” which failed to fully pass verification upon initial generation, indicating certain limitations of GPT-4 in generating complex interactive logic.

According to Fig. 2, in terms of development efficiency, ChatGPT-4 significantly reduced development time. Compared to traditional manual development, the development time for each functional module was greatly decreased. For example, when generating the “Student Entity (SE)” and “Three-Tier Architecture (TTA)” modules, ChatGPT-4 took only 0.8 hours and 0.4 hours, respectively, while manual development required approximately 1.5 hours and 3.0 hours. Notably, for the “Three-Tier Architecture” module, GPT-4 reduced the development time by 73%. However, for the complex “Page Navigation Button (PNB)” module, although the time taken for ChatGPT-4 to generate the code was 1 hour, an additional 0.5 hours was needed for manual optimization to ensure the logical correctness of page navigation. This further illustrates that GPT-4 still requires human intervention for optimal results in complex page interactions.

Combining the data from Table 1 and Fig. 2, there is a relatively consistent trend in functional correctness and usability. For simpler functional points such as “Login Page (LP)” and “Student Management Page (SM),” ChatGPT-4 was able to efficiently generate code while ensuring high correctness and usability. However, for complex functions like “Registration Backend Support (RBS)” and “Registration Page (RP),” although ChatGPT-4 generated code quickly, the functional correctness was lower, necessitating more manual optimization and debugging time.

Overall, ChatGPT-4 demonstrated outstanding performance in the efficiency of code generation, especially in significantly saving time in the development of basic functional modules. For instance, when generating simple structures like the “Student Entity” and “Three-Tier Architecture,” its speed far exceeded that of manual development, showcasing exceptional efficiency. However, when facing complex interactive logic and backend business processing, such as “Registration Backend Support” and “Page Navigation Button,” the generated code still required manual optimization and correction by developers to ensure functional correctness and the integrity of the system. This indicates that while ChatGPT-4 can effectively accelerate the development process, it still relies on a degree of human intervention in more complex scenarios.

5 Conclusion

The general-purpose large model represented by ChatGPT-4o demonstrates strong capabilities in code understanding and generation; however, it still falls short in generating complex practical software. To assess the capabilities of large models in this regard, this paper selects a typical practical web management system and conducts a comprehensive evaluation of ChatGPT-4o’s code generation abilities from the perspectives of code quality, structural clarity, and maintainability, utilizing a generic prompt engineering approach and manual optimization methods.

The experiments indicate that ChatGPT-4o exhibits robust code generation capabilities, efficiently completing the implementation of basic front-end pages and back-end logic, while also providing database interaction support. The code it generates saves developers a significant amount of time, particularly in writing foundational code, thus greatly enhancing development efficiency. However, as the complexity of system interaction logic and user experience increases, the limitations of ChatGPT-4o become more apparent. Certain details, such as page navigation logic and user experience optimization, still require human intervention to achieve optimal results.

In the future, with the continuous advancement of large model technology, we anticipate that ChatGPT-4o will further improve its performance in areas such as complex interaction logic, automatic optimization, and performance enhancement, bringing more possibilities for intelligent development.

Acknowledgments. This work is supported in part by Open Foundation of Hubei Key Laboratory of Applied Mathematics (Hubei University) (No. HBAM202304), Scientific and Technology Research Project of Hubei Provincial Education Department (No. Q20233007), and Scientific Research Foundation of Hubei University of Education for Talent Introduction (No. ESRC20230008).

References

[1] Gu, Q. (2023). LLM-based code generation method for Golang compiler testing. In Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 2201-2203).

[2] Wang J, Chen Y. A Review on Code Generation with LLMs: Application and Evaluation[C]//2023 IEEE International Conference on Medical Artificial Intelligence (MedAI). IEEE, 2023: 284-289..

[3] Koziolek H, Koziolek A. Llm-based control code generation using image recognition[C]//Proceedings of the 1st International Workshop on Large Language Models for Code. 2024: 38-45.

[4] Koziolek H, Grüner S, Hark R, et al. LLM-based and retrieval-augmented control code generation[C]//Proceedings of the 1st International Workshop on Large Language Models for Code. 2024: 22-29.

[5] Mu, F., Shi, L., Wang, S., et al. (2024). ClarifyGPT: A framework for enhancing LLM-based code generation via requirements clarification. Proceedings of the ACM on Software Engineering, 1(FSE), 2332-2354.

[6] Liu J, Xia C S, Wang Y, et al. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation[J]. Advances in Neural Information Processing Systems, 2024, 36.

[7] Liu, Z., Tang, Y., Luo, X., et al. (2024). No need to lift a finger anymore? Assessing the quality of code generation by ChatGPT. IEEE Transactions on Software Engineering.

[8] Tóth, R., Bisztray, T., & Erdődi, L. (2024). LLMs in web development: Evaluating LLM-generated PHP code unveiling vulnerabilities and limitations. In International Conference on Computer Safety, Reliability, and Security (pp. 425-437). Cham: Springer Nature Switzerland.

[9] Jamdade, M., & Liu, Y. (2024). A pilot study on secure code generation with ChatGPT for web applications. In Proceedings of the 2024 ACM Southeast Conference (pp. 229-234).

[10] Castelberg D, Flury F. LLM Assisted Development[D]. OST Ostschweizer Fachhochschule, 2024.
[11] Kazemitabaar M, Hou X, Henley A, et al. How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment[C]//Proceedings of the 23rd Koli Calling International Conference on Computing Education Research. 2023: 1-12.
[12] Wang J, Chen Y. A Review on Code Generation with LLMs: Application and Evaluation[C]//2023 IEEE International Conference on Medical Artificial Intelligence (MedAI). IEEE, 2023: 284-289..
[13] Voronin, D. N. (2024). Development and evaluation of an LLM-based tool for automatically building web applications (Doctoral dissertation, Massachusetts Institute of Technology).
[14] Calò T, De Russis L. Leveraging large language models for end-user website generation[C]//International Symposium on End User Development. Cham: Springer Nature Switzerland, 2023: 52-61.
[15] Cambaz D, Zhang X. Use of AI-driven Code Generation Models in Teaching and Learning Programming: a Systematic Literature Review[C]//Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1. 2024: 172-178.