A Study of Coding Framework Generation by ChatGPT

Haoming Zhu[1,a,*]

[1]Shandong University, No. 27, Shanda South Road, Jinan City, Shandong Province, China
a. 202300130137@mail.sdu.edu.cn
**corresponding author*

Abstract: In recent years, large language models (LLMs) have demonstrated remarkable capabilities in the field of code generation. However, existing research has primarily focused on algorithmic problem-solving code generation, with limited attention to the ability to generate framework code used in actual software development. Programming frameworks are vital tools in software development, effectively reducing development time and enhancing code compatibility. This paper takes the Qt framework in C++ as an example to systematically evaluate ChatGPT's performance in code generation at different levels of granularity (project-level, class-level, and function-level). To this end, we designed a test dataset (comprising 10 code generation projects of varying complexity) to assess the model's performance in terms of correctness, robustness, and user experience. In this process, we employed prompt engineering methods to ensure fair conversion. The experimental results show that while ChatGPT is capable of generating functional code in most cases, its performance in correctness, robustness, and user experience decreases as task complexity and code granularity increase. Nonetheless, with manual intervention or more detailed prompts, these issues can be largely resolved. Overall, ChatGPT shows potential in framework code generation, particularly for small to medium-sized tasks. This study reveals both the potential and limitations of LLMs in framework development, providing valuable insights for future improvements and applications.

Keywords: ChatGPT, Coding Framework, LLM, Code Generation.

1. Introduction

Nowadays, artificial intelligence, particularly large language models (LLMs), has demonstrated remarkable capabilities and potential in various fields. Leveraging the Transformer architecture, they have achieved near-human performance in natural language processing [1-2]. Among these fields, programming has emerged as one of the most prominent directions, showing rapid progress. With the help of LLMs, users can interact through natural language conversations and obtain the desired content. Currently, mainstream LLMs (such as ChatGPT) have achieved outstanding results in solving programming problems of varying complexity and across diverse application scenarios. When problems are described in natural language, LLMs can comprehend the essence of programming issues and exhibit strong generalization capabilities, allowing them to handle tasks beyond their training data [3-5]. These developments provide significant support and assistance to developers.

In actual project development, developers typically use programming frameworks rather than directly coding in programming languages. Programming frameworks provide developers with a predefined platform that offers a standardized structure and a toolkit. The use of programming frameworks can significantly improve development efficiency, code quality, and project maintainability, effectively reducing development cycles and enhancing code compatibility [6]. Programming frameworks play a crucial role in practical applications.

At present, research on the code generation capabilities of LLMs has primarily focused on their ability to solve algorithmic problems, assessing aspects such as correctness, complexity, and security [3-5][7-9]. While these algorithmic tasks have clear goals and the generated code is relatively straightforward, such studies have limited applicability to real-world software development. For developers, the use of programming frameworks is a key factor affecting development efficiency, compatibility, and collaboration, making it an essential component of project development. Therefore, evaluating LLMs' ability to generate code within programming frameworks is a necessary step toward transitioning cutting-edge technology into practical applications, addressing an urgent need for project developers.

This paper systematically investigates and analyzes the performance of large models in the task of generating Qt framework code in C++. First, existing research on code generation largely focuses on achieving specific and singular functionalities, with little exploration of the value of LLMs in real development applications [3-5]. Second, we selected 10 representative framework programming tasks from different perspectives and levels of difficulty, conducting code generation and conversion according to different levels of code granularity (function, interface, and overall project). Different levels of code granularity correspond to varying degrees of developer involvement relative to the output of large models. As the granularity decreases, the level of developer participation increases. By employing this approach, we can evaluate the generation performance of large models under varying degrees of involvement. During this process, we used identical prompts to ensure the fairness of the generated results (which needs to be elaborated). Subsequently, we tested the functionality and quality of the generated code to assess its effectiveness at different levels of granularity.

The experiments demonstrate that large model systems, such as ChatGPT, are generally effective in generating framework code. At different levels of granularity, they can essentially produce the required code. However, as the granularity increases, the proportion and number of syntax errors in the code also increase. Nevertheless, after manual modifications or additional prompts, the models can typically meet the required functionality in a more complete manner.

2. Background

2.1. LLM and ChatGPT

Large models represent the most advanced technologies in the field of artificial intelligence today. They are highly complex neural network models built on deep learning and natural language processing technologies [10]. ChatGPT, as one of the representative products of these models, is renowned for its powerful language understanding and generation capabilities [2].

ChatGPT's ability to comprehend and generate human language stems from its training process. It is trained on vast amounts of textual data, allowing it to learn the complex relationships between syntax, semantics, and context, enabling it to effectively process and generate natural language text. This capability has shown significant potential and practicality across various application scenarios.

In practical applications, large models like ChatGPT can be used for numerous tasks and industries. In the field of natural language processing, they can be applied to tasks such as text generation, machine translation, sentiment analysis, and intelligent customer service [1-2][11-12]. For example, ChatGPT can serve as an intelligent assistant or virtual customer service system, capable of understanding users' questions and providing corresponding answers and solutions, thus improving user experience and service efficiency. In the medical field, these models can assist doctors in diagnostic reasoning and suggesting treatment plans, help increase research writing efficiency, and accelerate drug development [13]. In the legal sector, they can assist lawyers in drafting legal documents and offering legal advice [14]. Additionally, large models can also undertake tasks in software development, such as code generation, code analysis, and code testing [3-5][15].

2.2. Survey on Framework-level Code Generation

Current research on the code generation capabilities of LLMs mainly focuses on their ability to solve algorithmic problems, limiting the tasks to relatively simple and well-defined code generation. For instance, D. Yan et al.[5,7] conducted empirical studies using large datasets to evaluate the correctness and quality of generated code, such as the algorithm code generation on the LeetCode platform. R. Khoury et al.[16] generated 21 programs in different programming languages and prompted ChatGPT to identify and fix security vulnerabilities in the code, assessing the security of the generated code. D. Justin et al. [17-18] compared human-translated code with AI-translated code to evaluate ChatGPT's potential in code translation.

However, such work does not consider the direct generation of framework code, such as Qt, using large models, nor does it examine code generation performance across different levels of granularity and perspectives. As a result, the evaluations tend to be incomplete and vague, providing insufficient insight into the practical application of these models.

3. Approach

3.1. Programming Tasks

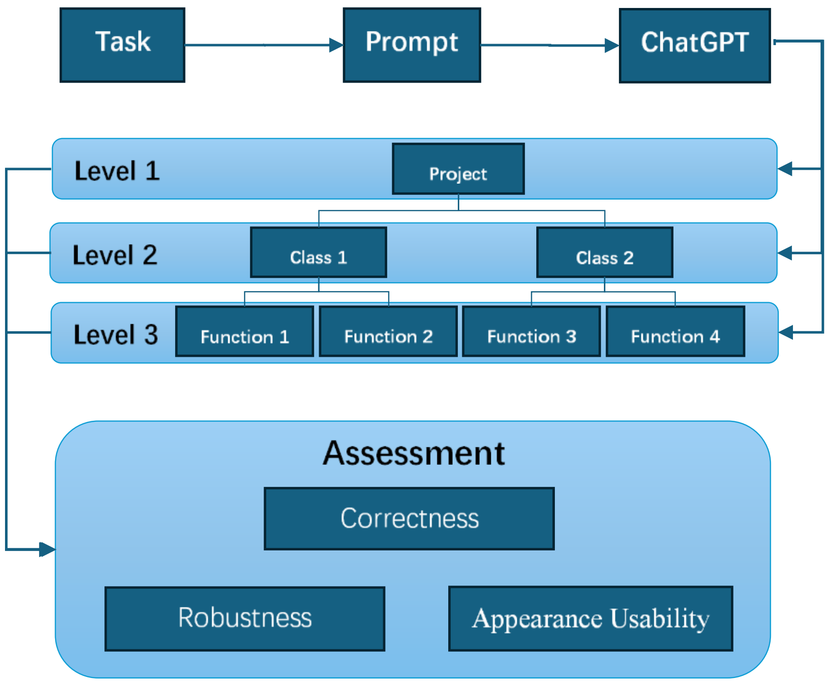

Figure 1: Workflow of LLM-based Code Framework Transformation.

In the experiment, We used prompt engineering [19-20] to break down complex programming tasks into specific prompts that are easier for ChatGPT to understand and handle, improving its comprehension and execution of tasks. We divided the programming tasks into three levels: the overall project level, the class level, and the function level.

As we gradually refined and decomposed the entire project, ChatGPT’s involvement in project development decreased, while the developer’s participation increased. Finally, we independently evaluated the results at each level, assessing their performance in the following areas: code correctness, robustness of exception handling, and user experience. These evaluations allow us to comprehensively measure the impact of prompt engineering at different levels on the quality of project development.

3.2. Prompt Engineering

To generate the framework code fairly and comprehensively, we have designed specialized prompt templates from different levels as follows.

3.2.1. Overall Project

You are a software developer. Please use the Qt framework and C++ language to write a restaurant management system program. The program must allow administrator login and user login, storing information in `.dat` files. Different login identities should enable different functionalities. The program needs to manage four `.dat` files: administrator information, user information, dish information, and feedback information. Administrator information should include account and password; user information should include name, account, and password; dish information should include dish name, ingredients, supply location, and price; feedback information should include dish name and feedback content. When logged in as an administrator, the program should have functions such as adding administrators, deleting administrators, modifying administrator passwords, adding dish information, deleting dish information, viewing dish feedback, adding users, and deleting users. When logged in as a user, the program should allow viewing dish information and providing dish feedback.

3.2.2. Class Level

You are a software developer. Please use the Qt framework and C++ language to create a visual restaurant management system. Follow my guidance step by step to complete the project.

We need you to generate the user interfaces one by one. First, create the main interface that implements administrator login and user login. Different login identities should redirect to different interfaces to enable various functionalities. Administrator information should include account and password; user information should include name, account, and password. Store the information in two `.dat` files. The generated code should include all necessary files, including the UI files.

Now implement the administrator functionality. When logged in as an administrator, the system should provide functions such as adding administrators, deleting administrators, modifying administrator passwords, adding dish information, deleting dish information, viewing dish feedback, adding users, and deleting users. Dish information should include dish name, ingredients, supply location, and price. Feedback information should include dish name and feedback content. Store the information in `.dat` files. All functionalities should be integrated into a single interface.

Now complete the user functionality. When logged in as a user, the system should allow viewing dish information and providing feedback on dishes.

3.2.3. Function Level

You are a software developer. Please use the Qt framework and C++ language to create a visual restaurant management system. Follow my guidance step by step to complete the project.

Write a slot function for the interface class that completes all the following functionalities. [Specific function details]

(For example: "Read administrator data from `admins.dat` file. Administrator information should include account and password. Compare with the `lineEdit` (account) and `lineEdit_2` (password) fields in the UI file. If the login is successful, leave a redirect interface; if it fails, display an error message.")

3.3. Multi-level Object Design

At the overall project level, we provide ChatGPT with a detailed and comprehensive description, covering all functional requirements of the project. The goal is to enable ChatGPT to fully understand and independently complete the development of the entire project. The prompts at this level are usually long and require ChatGPT to generate complete code or scripts.

At the class level, we further break down the overall functionality of the project into separate interface modules. Each module represents a specific part of the project's functionality. We provide ChatGPT with a description of the specific functionalities required for each interface, and it generates the corresponding code. At this stage, the developer's involvement starts to increase, as they are responsible for integrating the individual interface codes into the project.

At the function level, we further refine each interface's functionality into multiple specific functions. Each function corresponds to a particular sub-functionality. We provide ChatGPT with detailed descriptions of each function, asking it to generate the code that implements the required functionality. The prompts at this level are usually shorter and more specific, and the developer's involvement increases significantly, as they are responsible for integrating the functions into the interface and further optimizing the code.

4. Evaluation

4.1. Datasets

In this study, I designed a series of datasets to evaluate and test the code generation capabilities of large language models (LLMs) under the Qt framework. These datasets include 10 framework code generation projects with varying levels of complexity.

Table 1: Evaluation System Requirement Dataset.

System Requirement Name	
Calculator	To-do List
Simple Database Management System	Restaurant Management System
Personal Finance Management System	Vehicle Rental System
Online Examination System	Inventory Management System
Project Management System	Library Management System

Each project is further divided into different code granularity levels to simulate the various levels and requirements of actual software development. The code granularity division aims to cover all aspects, from high-level design to low-level implementation. This step-by-step refinement effectively evaluates the model's performance at different programming depths, ensuring the fairness, rationality, and objectivity of the experiment.

The principles of code granularity division are based on the following considerations:

Project Level: The model is required not only to generate code and implement functionality but also to demonstrate project management capabilities, including file management and module coordination.

Class Level: The model is expected to divide functionalities into various functions for implementation.

Function Level: The model must handle specific programming tasks, such as data processing and logical operations.

By structuring the levels in this way, we can thoroughly analyze the model's performance in different programming activities and thus gain a more comprehensive understanding of its potential applications in real-world software development.

4.2. Metrics

In evaluating code quality, we used the following primary evaluation metrics: code correctness, robustness in exception handling, and the aesthetic quality of the interface design. To achieve a quantitative assessment, the following evaluation methods are proposed:

1) Correctness: The correctness of the code refers to whether the program executes as expected and produces the correct results. To assess this, we can divide the overall code into smaller code testing units and test each one. Correctness is calculated using the following formula, where M_p represents the number of tests passed and M is the total number of tests:

$$C = \frac{M_P}{M} \tag{1}$$

2) Robustness: Robustness refers to the program's ability to remain unaffected or recover well from extreme situations. It is calculated as follows, where L_p represents the number of tests passed under extreme conditions, and L is the total number of extreme tests:

$$R = \frac{L_P}{L} \tag{2}$$

3) Appearance Usability: The appearance usability of the interface design is measured by user satisfaction with the UI design, with a score range of [0, 1]. To do this, we surveyed N users who rated the generated interface, with the score given by the i-th user recorded as S_i. The aesthetic usability is calculated as:

$$E = \frac{\sum_N s_i}{N} \tag{3}$$

4.3. Experimental Setup

In this experiment, we used the ChatGPT-4o model to generate code and utilized Qt 6.7.1 and Qt Creator 14.0.1 as the experimental environment. The compiler version was MinGW 11.2.0 64-bit. During the experiment, for the correctness evaluation, we tested each unit with 10 test cases. For the robustness evaluation, we set up 8 extreme environments, including excessive input length, multiple concurrent operations, non-compliant operations, and resource-intensive scenarios. For the aesthetic usability evaluation, we recruited 5 students to rate each Qt interface generated by the model.

4.4. Results

Table 2: Result Statistics.

System Requirement Name	Function			Class			Project		
	One	Two	Three	One	Two	Three	One	Two	Three
Calculator	1	0.87	0.98	1	0.75	0.84	0.9	0.53	0.48
To-do List	1	0.68	0.95	0.95	0.8	0.87	0.85	0.57	0.52
Simple Database Management System	0.98	0.83	0.81	0.91	0.66	0.57	0.84	0.43	0.63
Restaurant Management System	0.96	0.64	0.88	0.93	0.61	0.82	0.72	0.54	0.43
Personal Finance Management System	0.97	0.75	0.94	0.85	0.55	0.72	0.71	0.37	0.37
Vehicle Rental System	0.92	0.63	0.92	0.88	0.52	0.68	0.69	0.33	0.54
Online Examination System	0.94	0.67	0.88	0.79	0.46	0.76	0.77	0.27	0.4
Inventory Management System	0.92	0.72	0.93	0.82	0.49	0.61	0.82	0.29	0.47
Project Management System	0.95	0.66	0.92	0.87	0.51	0.78	0.76	0.36	0.61
Library Management System	0.96	0.76	0.91	0.83	0.54	0.8	0.81	0.41	0.38
Total	0.96	0.72	0.91	0.88	0.59	0.75	0.79	0.41	0.48

Note. Correctness = One; Robustness = Two; Appearance Usability = Three.

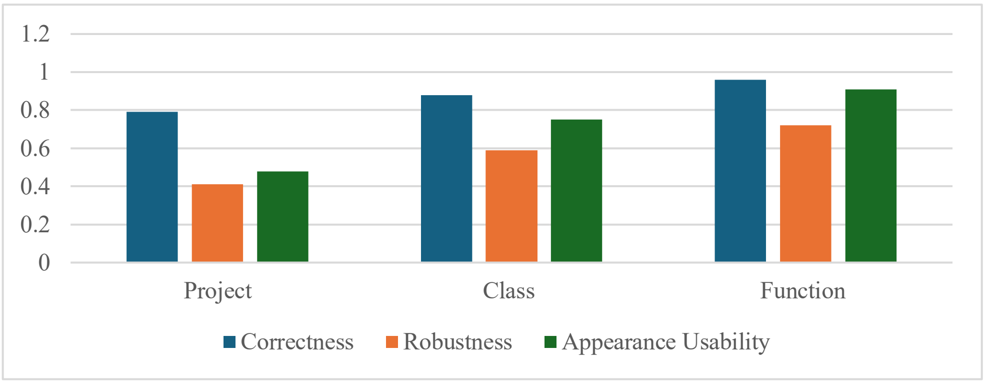

Figure 2: Result Statistics.

Correctness: ChatGPT exhibited a relatively consistent high level of performance regarding code correctness across all three levels. This suggests that it can generate the correct logic for most code during the generation process. However, minor errors, such as forgetting to include necessary header files, are still challenging to avoid. Notably, correctness slightly declined at the class and project levels, likely due to the increased complexity of the code in each generation. Quantitative results showed average correctness scores of 0.96 at the function level, 0.88 at the class level, and 0.79 at the project level. Making manual corrections in error instances, or with additional prompting, these issues were often resolved.

Robustness: ChatGPT's performance in managing extreme cases was relatively good at the function level, but overall, it was still unsatisfactory. The robustness score fell from 0.72 at the function level to 0.41 at the project level, indicating that the code produced struggles with complex scenarios in real-world applications. Specifically, scenarios involving concurrent operations or high memory usage led to failures at the class and project levels. These findings suggest that while ChatGPT can generate functional code, additional safeguards and manual optimizations are necessary to ensure robustness in extreme conditions.

Appearance Usability: In terms of usability and appearance design, ChatGPT performed well at the function and class levels but poorly at the project level. Usability scores were 0.91, 0.75, and 0.48 for the function, class, and project levels, respectively. This indicates that maintaining consistent interface appearance becomes increasingly challenging as project complexity rises. Survey participants noted that project-level interface designs were often cluttered, with inconsistencies in layout and style. This underscores the difficulty of ensuring high-quality interface design without human intervention as the scope and granularity of the project expand.

Nevertheless, the experimental results reveal some shortcomings. First, the experiment focused primarily on the Qt framework and did not cover other programming frameworks or languages, limiting the generalizability of the results. Second, although the evaluation metrics included correctness, robustness, and aesthetic usability, other potential dimensions—such as maintainability, scalability, and runtime efficiency—were not considered. Additionally, although prompt engineering was employed, further exploration is needed to determine if the prompt design could better tap into the model's potential. In conclusion, while large language models (LLMs) still have shortcomings in code generation, their potential to improve development efficiency and reduce costs cannot be ignored, offering new perspectives for future software development practices.

5. Conclusion

This study closely examined the transformation capability of ChatGPT from general language like C++ to the common-used code framework like the Qt. We broke down programming tasks into three levels: project, class, and function, highlighting both the strengths and weaknesses of ChatGPT and design specific prompt engineering template for few-shot guidance. The experiment showed that ChatGPT performed really well with simpler tasks—especially at the function level—where it mostly produced correct and effective code. However, as the complexity ramped up to class and project levels, errors like syntax issues and missing dependencies started to increase. This indicates that while LLMs like ChatGPT can assist with software development, they still need quite a bit of human participation when it comes to larger, more complex projects.

In future work, this assessment could extend to other programming frameworks and languages to see how generalizable LLMs are in different development contexts. Also, optimizing prompt engineering methods could lead to even better code quality and robustness.

References

[1] Minaee, S., Mikolov, T., Nikzad, N., et al. (2024). Large language models: A survey. arXiv preprint arXiv:2402.06196.

[2] Chang, Y., Wang, X., Wang, J., et al. (2024). A Survey on Evaluation of Large Language Models. ACM Trans. Intell. Syst. Technol. 15, 3, Article 39 (June 2024), 45 pages. https://doi.org/10.1145/3641289

[3] Sakib, F. A., Khan, S. H., & Karim, A. H. M. (2023). Extending the frontier of ChatGPT: Code generation and debugging. arXiv preprint arXiv:2307.08260.

[4] Frank F. Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. (2022). A systematic evaluation of large language models of code. In Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming (MAPS 2022). Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3520312.3534862

[5] Yan, D., Gao, Z., & Liu, Z. (2023). A closer look at different difficulty levels code generation abilities of ChatGPT. In 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 1887-1898). Luxembourg, Luxembourg. https://doi.org/10.1109/ASE56229.2023.00096.

[6] Ruijie Pang, "Analysis of Python web development applications based on the Django framework, " Proc. SPIE 13181, Third International Conference on Electronic Information Engineering, Big Data, and Computer Technology (EIBDCT 2024), 131816C (19 July 2024); https://doi.org/10.1117/12.3031411

[7] Liu, Z., Tang, Y., Luo, X., Zhou, Y., & Zhang, L. F. (2024). No need to lift a finger anymore? Assessing the quality of code generation by ChatGPT. IEEE Transactions on Software Engineering, 50(6), 1548-1584. https://doi.org/10.1109/TSE.2024.3392499.

[8] M. Liu, T. Yang, Y. Lou, X. Du, Y. Wang and X. Peng. (2023). CodeGen4Libs: A Two-Stage Approach for Library-Oriented Code Generation. In 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 434-445), Luxembourg, Luxembourg, https://doi.org/10.1109/ASE56229.2023.00159.

[9] Siddiq, M. L., Majumder, S. H., Mim, M. R., Jajodia, S., & Santos, J. C. S. (2022). An empirical study of code smells in transformer-based code generation techniques. In 2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM) (pp. 71-82). Limassol, Cyprus. https://doi.org/10.1109/SCAM55253.2022.00014.

[10] Vaswani A. Attention is all you need[J]. Advances in Neural Information Processing Systems, 2017.

[11] Jiao, W., Wang, W., Huang, J., et al. (2023). Is ChatGPT a good translator? Yes with GPT-4 as the engine. arXiv preprint arXiv:2301.08745.

[12] Sahari, Y., Al-Kadi, A.M.T. & Ali, J.K.M. A Cross Sectional Study of ChatGPT in Translation: Magnitude of Use, Attitudes, and Uncertainties. J Psycholinguist Res 52, 2937–2954 (2023). https://doi.org/10.1007/s10936-023-10031-y

[13] Sallam, M. (2023). ChatGPT utility in healthcare education, research, and practice: Systematic review on the promising perspectives and valid concerns. In Healthcare (Vol. 11, No. 6, p. 887). MDPI. https://doi.org/10.3390/healthcare11060887.

[14] Choi, J. H., Hickman, K. E., Monahan, A. B., et al. (2021). ChatGPT goes to law school. Journal of Legal Education, 71, 387.

[15] Yi, G., Chen, Z., Chen, Z., Wong, W. E., & Chau, N. (2023). Exploring the capability of ChatGPT in test generation. In 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRS-C) (pp. 72-80). Chiang Mai, Thailand. https://doi.org/10.1109/QRS-C60940.2023.00013.

[16] Khoury, R., Avila, A. R., Brunelle, J., & Camara, B. M. (2023). How secure is code generated by ChatGPT? In 2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC) (pp. 2445-2451). Honolulu, Oahu, HI, USA. https://doi.org/10.1109/SMC53992.2023.10394237.

[17] Yan, W., Tian, Y., Li, Y., et al. (2023). Codetransocean: A comprehensive multilingual benchmark for code translation. arXiv preprint arXiv:2310.04951.

[18] Justin D. Weisz, Michael Muller, Steven I. Ross, Fernando Martinez, Stephanie Houde, Mayank Agarwal, Kartik Talamadupula, and John T. Richards. 2022. Better Together? An Evaluation of AI-Supported Code Translation. In Proceedings of the 27th International Conference on Intelligent User Interfaces (IUI '22). Association for Computing Machinery, New York, NY, USA, 369–391. https://doi.org/10.1145/3490099.3511157

[19] White, J., Fu, Q., Hays, S., et al. (2023). A prompt pattern catalog to enhance prompt engineering with ChatGPT. arXiv preprint arXiv:2302.11382.

[20] Ratnayake, H., Wang, C. (2024). A Prompting Framework to Enhance Language Model Output. In: Liu, T., Webb, G., Yue, L., Wang, D. (eds) AI 2023: Advances in Artificial Intelligence. AI 2023. Lecture Notes in Computer Science(), vol 14472. Springer, Singapore. https://doi.org/10.1007/978-981-99-8391-9_6