A Novel Method of PLC Code Generation based on Large Language Models

Jiatong Zheng
University of Chinese Academy of Sciences, Beijing100049, China
State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Science, Shenyang 110016, China
Shenyang, China
zhengjiatong1@sia.cn

Dong Li
University of Chinese Academy of Sciences, Beijing100049, China
State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China
Shenyang, China
lidong@sia.cn

Yong Sun
University of Chinese Academy of Sciences, Beijing100049, China
State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Science, Shenyang 110016, China
Shenyang, China
sunyong@sia.cm

Chunhe Song
University of Chinese Academy of Sciences, Beijing100049, China
State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Science, Shenyang 110016, China
Shenyang, China
* Corresponding author: songchunhe@sia.cn

***Abstract*—At present, large language models are very important and can help solve problems in fields with strong versatility. However, in some industrial control fields, such as the programmable logic controller (PLC) field, the versatility of large language models cannot solve the problem of compiling platform code application well, as the code generation problem in the PLC field is difficult because the code adaptability of different software in the PLC field is not good. This paper proposes a novel method of PLC code generation based on Large Language Models. First, a PLC-100 industrial dataset is constructed, which contains 10 types of 100 prompt words. It covers all aspects of structured text content and can be used as a high-quality prompt set. Second, a PLC code generation method based on GPT-4 with an adaptive evaluation system is proposed using the dataset. The experimental results show that, the dataset and the proposed evaluation algorithm in this paper can be well applied to large model code generation in the field of industrial control.**

Keywords- Control Logic Generation; Large Language Models; IEC 61131-3; Adaptive feekback

I. INTRODUCTION

In recent years, the breakthrough progress of large models in the field of code generation has enabled deep learning to show significant advantages in the task of converting natural language to code. Early applications of sequence-to-sequence (Seq2Seq) models [1] showed that deep learning can effectively convert text descriptions into code, significantly reducing the burden on developers. Subsequently, models based on abstract syntax trees (ASTs) [2] further enhanced the grammatical consistency[3] and quality of generated code[4]. The introduction of pre-trained models, such as CuBERT [5] and CodeBERT [6], has achieved good results in multiple programming languages, and GraphCodeBERT [7] has improved code comprehension capabilities by using data flow and structural information. Large-scale pre-trained models such as GPT-3 [8] , Copilot [9] and GPT-4 [10] provide strong support for general code generation, but their performance in specific fields such as industrial control still needs to be verified. In addition, researchers explored improvements in prompt engineering [12,13] and multi-round dialogue generation mode [14], further improving the accuracy of code generation. However, applications in specific fields such as industrial automation still need more exploration, and this article will conduct further research based on existing results.

In response to the above problems, the contributions of this paper can be summarized as follows:

First, a PLC-100 industrial dataset is constructed. It contains 10 types of 100 prompt words, which covers all aspects of structured text content and can be used as a high-quality prompt set. This prompt set lays the foundation for a more formal LLM quality evaluation benchmark in the future, especially for the comparison and improvement of code generation effects.

Second, the PLC code generation capabilities of large models such as GPT-4o are evaluated through the Human-Based Evaluation Scores Algorithm. An adaptive evaluation scheme is designed. After the large model platform code is generated, it is reproduced through software. After reproduction, the quality effect of code generation is judged by evaluation indicators, and the algorithm of traditional evaluation indicators is innovated.

Third, an adaptive evaluation indicator method is proposed to solve the limitation of traditional evaluation indicators. In terms of the algorithm of evaluation indicators, we added adaptive algorithms to the BLEU series and pass@k series indicators, which greatly improved the algorithm efficiency and pointed out code problems in a timely manner, making the algorithm evaluation system more meaningful; the code generation capability of large models was evaluated using the

Human-Based Evaluation Scores Algorithm, and the generated code was scored to ensure that the code quality was feasible and highly operable.

II. METHODOLOGY

A. Prompt Collection

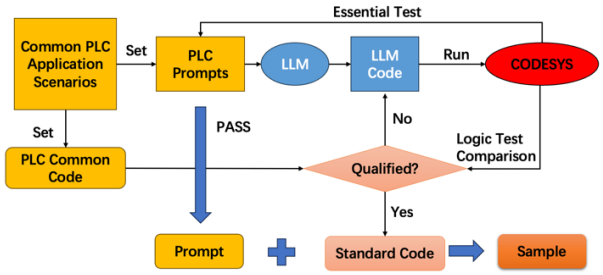

Figure 1: PLC-100 Prompt Words Collection Process

The Figure 1 shows the overall logical flow of the prompt set production, which aims to evaluate the effectiveness and accuracy of the PLC code generated based on the large language model (LLM). Through this process, the quality of PLC code generated based on LLM can be systematically verified and improved, while promoting its practical application in industrial automation scenarios.

B. Code Generation Based on Large Language Model

We summarized and analyzed 100 answers from ChatGPT on Github. The 100 data were divided into ten categories, with 10 data in each category. For example, for advanced process control, ChatGPT demonstrated the ability to generate complex dynamic models and model predictive control (MPC) codes. For example, in the prompt MPC MATLAB Distillation Column, it generated a dynamic model for a distillation column with several specific and reasonable process parameters. It utilized MATLAB solvers for solving ordinary differential equations (ODEs). In a follow-up prompt, we requested an MPC solution for this model, which ChatGPT provided as a basic implementation using the fmincon function. The generated code was successfully executed in MATLAB. The feedback from three MPC experts on the code was positive, but a more thorough evaluation was necessary. This category also includes responses to other control techniques, such as fuzzy logic control, artificial neural network control, and statistical process control, which were generated in Python, C++, or MATLAB as requested. For specific data classification details, please see the GitHub link in the lower left corner of the page header.

C. Human Based Evaluation Algorithm

Before using evaluation indicators to judge the quality of code generation, the code generated by the large model is first scored algorithmically. The Figure 2 proposes an algorithm to evaluate the quality of code generated by the model based on multi-dimensional criteria and generate a comprehensive score for each piece of code.

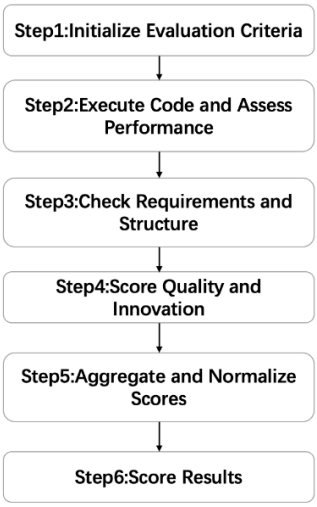

Figure 2: Algorithm Scoring Steps

First, the algorithm can be evaluated in multiple dimensions. The algorithm combines functional, structural, and qualitative analysis to provide a comprehensive code quality assessment. Second, it is scalable: it supports batch processing of multiple code segments and saves the results for further analysis. Third, it is customizable. The scoring criteria can be adjusted according to specific project or research needs. The specific code for the algorithm scoring can still be found in the GitHub link.

III. METRICS

Among the evaluation indicators of code generation, there are two commonly used evaluation indicator series: BLEU series and Pass@k series. The BLEU series of evaluation indicators was proposed by IBM to provide a fast and language-independent automatic evaluation method to replace expensive manual evaluation. Its core concept is that "the closer the machine translation is to professional human translation, the higher the quality." It evaluates the quality by calculating the degree of match between the candidate translation and the reference translation, with a score range of 0 to 1, where 1 indicates a perfect match. Pass@K is an evaluation indicator that is mainly used to evaluate the performance of code generation models, especially when dealing with programming problems. It measures whether the model can generate at least one correct code snippet for a given problem in K attempts, where correctness is usually verified by unit testing, and the score range is 0 to 1, where 1 indicates that all code snippets can be executed correctly. Both evaluation indicators have certain limitations when evaluating the quality of code generation. The evaluation of the BLEU series requires fixing the n in the n-gram in the formula to determine the score range of code generation, and the n value must be fixed each time the calculation is performed; the Pass@k series also needs to determine the score range by fixing the k value, which has certain limitations. Through the adaptive algorithm, different weights are given to different n and k values of BLEU and Pass@k, namely AdaptiveBLEU and AdaptivePass@k.

PLC-100 Dataset and Experiment :
https://github.com/ZzzJjt/PLC-100

Compared with the traditional BLEU and Pass@k methods, this method can more objectively reflect the quality of code generation. The following is a detailed analysis of AdaptiveBLEU and AdaptivePass@k.

A. Adaptive Evaluation Index

1) AdaptiveBLEU:

The traditional BLEU metric uses fixed n-gram weights, but different tasks pay different attention to n-grams. We can study an adaptive n-gram weight adjustment mechanism to enable the BLEU metric to dynamically adjust weights based on task or sentence characteristics. In this paper, we propose a method to achieve adaptive weight adjustment BLEU.

We define a new adaptive n-gram weight formula αn, whose weight depends on the level n of the n-gram and the sentence length difference.

$$\alpha_n = \frac{\gamma_n}{\sum_{n=1}^{N}\gamma_n}, \quad \gamma_n = \lambda \cdot \left(\frac{n}{N}\right)^{\beta} \cdot \left(\frac{\min(c,r)}{\max(c,r)}\right)^{\theta} \tag{1}$$

Among them, α_n is the adaptive weight of the n-gram. The sum is normalized to 1. γ_n is the basic weight, which controls the contribution of n-gram weights through different parameters. λ is a scaling parameter that adjusts the overall weight and can usually be set to 1. n: The level of the current n-gram, such as 1-gram, 2-gram, etc. N: The level of the maximum n-gram, such as 4 for 1-gram to 4-gram. β: A control parameter for the n-gram level, used to emphasize the influence of high n-grams (larger n) or low n-grams (smaller n). The BLEU formula with adaptive n-gram weights can be expressed as:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} \alpha_n \log p_n\right) \tag{2}$$

where BP is the shorthand length penalty, p_n is the precision of the n-gram, and α_n is the adaptive weight.

2) AdaptivePass@k

Currently, Pass@k usually uses a fixed number of candidate codes (the value of k is fixed, such as k=5 or k=10), but in actual application scenarios, some tasks may not need to generate so many candidate codes to find the correct solution, while other more complex tasks may require more candidate codes. We propose an adaptive candidate generation strategy that dynamically adjusts the value of k to optimize generation efficiency and resource usage. We define a formula to express the dynamically adjusted Pass@k as follows:

$$\text{AdaptivePass@k} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}(\{j\} \in [1, k(C_i)]) \tag{3}$$

where N is the total number of tasks; $k(C_i)$ is the number of candidates dynamically adjusted based on the complexity of the i-th task C_i ; 1 is an indicator function that takes the value 1 when at least one candidate solution is correct and 0 otherwise; candidate i, j is the j-th candidate solution for the i-th task; reference i is the reference correct solution for the i-th task.

IV. EXPERIMENTS

The platform and experimental environment of this experiment are based on CODESYS, an internationally commonly used industrial control software. CODESYS (Controller Development System) is an integrated development environment (IDE) widely used for PLC programming in the field of industrial automation based on the IEC 61131-3 standard. It supports multiple programming languages such as structured text, ladder diagram, function block diagram, etc., and can write, debug and deploy control logic. CODESYS provides cross-vendor soft PLC solutions, is compatible with multiple hardware platforms, and supports real-time task management, visualization, remote debugging, device communication and other functions. Due to its flexibility and wide range of device support, CODESYS occupies an important position in industrial control systems. Other compilation platforms such as python, c#, matlab, etc. are more common, so I will not go into details here. The version used by CODESYS is V3.5 SP13.3, which is compatible with the Windows operating system. Figure 3 shows the generated code and corresponding interface of "Car Wash Control" in PLC Programing Tasks in the PLC-100 dataset.

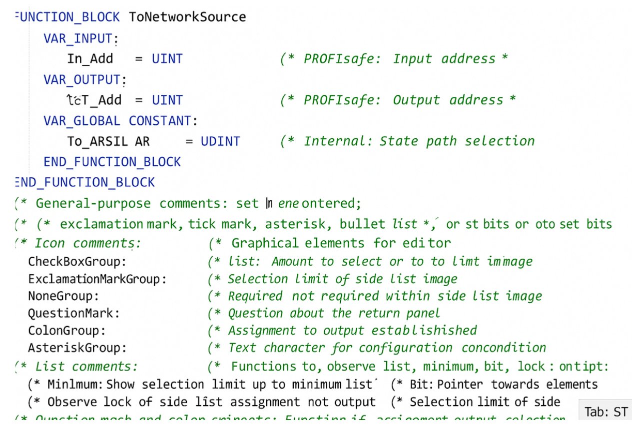

Figure 3: "Car Wash Control" Program in Codesys

Since 100 data are too redundant in the result display, the following operations are performed on the data of the BLEU series and pass@k series. For the convenience of display, 10 types of PLC-ST data are displayed, and each type originally contains 10 data. The following experiments will compare the numerical values of the traditional BLEU algorithm and the AdaptiveBLEU algorithm, and the comparison between the traditional Pass@k and AdaptivePass@k.

1) BLEUSeries:

In the BLEU-n (n=1,2,3,4) algorithm, the commonly used n-gram value is 4. The other n (n=1,2,3) may lead to inaccurate evaluation indicators due to n being too small, so the uncommon BLEU-n (n=1,2,3) is not mentioned here. Figure 4 show the experimental results of the BLEU series series. When the n-gram value is 4, the running results are consistent with the ideal expectations. When the Human-Based Evaluation Scores Algorithm score is in the low range, the value of Adaptive BLEU is slightly lower than most of the results of BLEU-4; when the Human-Based Evaluation Scores Algorithm score is in the middle range, the value of Adaptive BLEU is comparable to most of the results of BLEU-4; when the Human-Based Evaluation Scores Algorithm score is in the high range, the value of Adaptive BLEU is higher than most of the results of BLEU-4. This shows that the algorithm innovation of Adaptive BLEU can not only inherit the advantages of the BLEU series of indicators, but also further improve the evaluation ability of code quality by dynamically adjusting the weight and matching

PLC-100 Dataset and Experiment : https://github.com/ZzzJjt/PLC-100

method of n-gram. Especially in the code generation task, Adaptive BLEU more effectively balances vocabulary matching and semantic consistency, avoiding the limitations of relying solely on fixed n-gram. This also verifies the superiority of Adaptive BLEU in complex generation tasks, making it a more comprehensive and reliable indicator for measuring the quality of code generation. When the Human-Based Evaluation Scores Algorithm score is in the low segment, due to the limitations of the current general large model in the field of industrial control, the generated code may be significantly different from the traditional reference code. However, after the compensation of the innovative evaluation algorithm, the value of Adaptive BLEU can reflect the true quality of the generated code. After reproducing on codesys, the generation effect of the low segment is not feasible, ensuring the authenticity of the code generation.

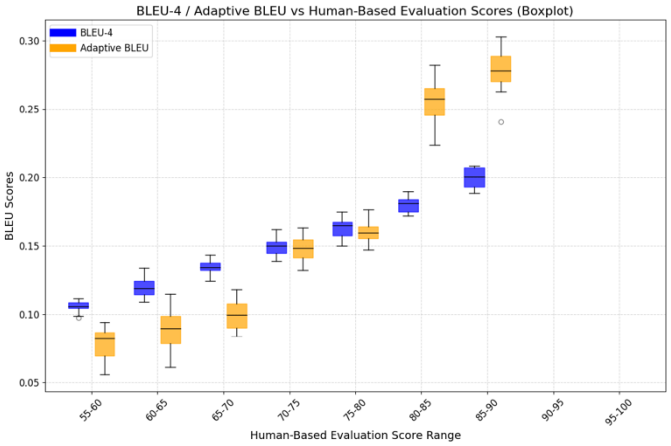

Figure 4: BLEU-4/AdaptiveBLEU vs Human-Based Evaluation Scores (Boxplot)

2) Pass@k Series:

Figure 5 and Figure 6 show the algorithm results of the Pass@k series. The picture shows the algorithm results of the Pass@k series. Pass@k directly displays all the results. From the chart, we can see that when the Human-Based Evaluation Scores Algorithm score is in the low segment, the value of Adaptive pass@k is slightly lower than most of the results of Pass@k (k=5, 10); when the Human-Based Evaluation Scores Algorithm score is in the middle segment, the value of Adaptive pass@k is equivalent to most of the results of Pass@k (k=5, 10); when the Human-Based Evaluation Scores Algorithm score is in the high segment, the value of Adaptive_pass@k is higher than most of the results of Pass@k (k=5, 10). This shows that the algorithm of Adaptive_pass@k can also be used for subsequent code evaluation in the industry.

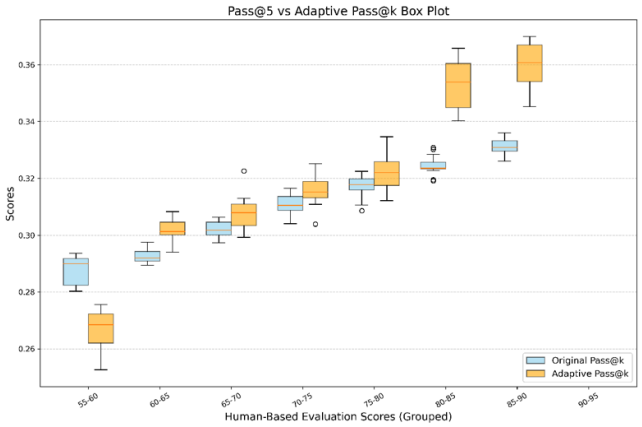

Figure 5: Pass@5 vs Adaptive Pass@k BoxPlot

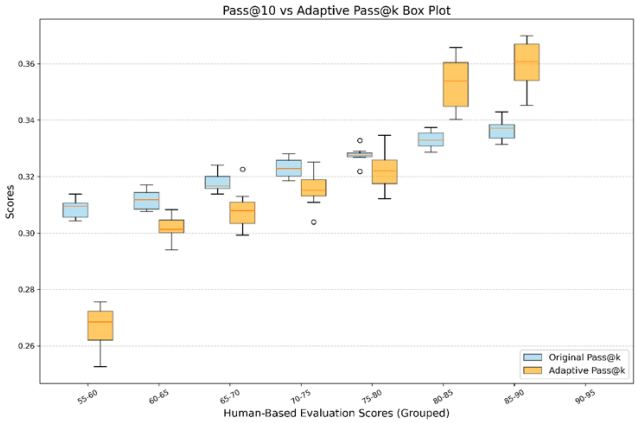

Figure 6: Pass@10 vs Adaptive Pass@k BoxPlot

According to the generated chart, we can also see some points of pass@k. When the Human-Based Evaluation Scores Algorithm score is in the low segment, the value of Adaptive_pass@k is slightly lower than most of the results of Pass@k (k=5, 10). This also reflects the limitations of large models in the field of industrial control to some extent. After the compensation of innovative algorithm evaluation, Adaptive_pass@k can also feedback the real quality of the generated code.

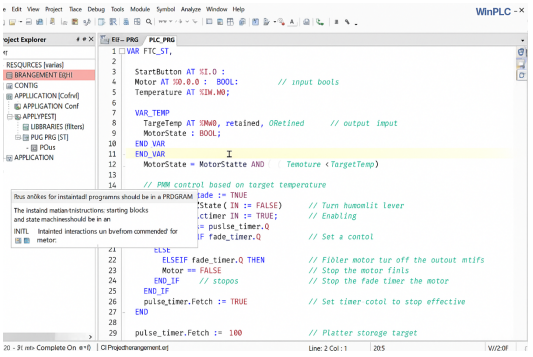

Figure 7 : “Car Wash Control” Programing Results

The scored high-quality code is verified on the industrial software codesys. From the experimental results, the compiled

PLC-100 Dataset and Experiment :
https://github.com/ZzzJjt/PLC-100

code cannot be directly used for copying and pasting for the time being. This is because the large model code generation cannot be fully transplanted in a specific usage environment. It is still necessary to manually input information for the actual IO usage interface, and the generated code has certain warnings. This is where the generated code still needs to be improved, as shown in Figure 7. The generated code can be successfully run on codesys after manual input of the IO port, proving that the quality of the generated code meets the IEC-61131-3 standard. After certain processing, it can be used for industrial code control.

V. Conclusions

The breakthrough progress of large models in the field of code generation has enabled deep learning to show significant advantages in the task of converting natural language to code. However, applications in specific fields such as industrial automation still need more exploration, and this article will conduct further research based on existing results. In this paper, first, a PLC-100 industrial dataset is constructed, which lays the foundation for a more formal LLM quality evaluation benchmark in the future, especially for the comparison and improvement of code generation effects. Second, the PLC code generation capabilities of large models such as GPT-4o are evaluated through the Human-Based Evaluation Scores Algorithm, and an adaptive evaluation scheme is designed. Third, an adaptive evaluation indicator method is proposed to solve the limitation of traditional evaluation indicators, and the code generation capability of large models was evaluated using the Human-Based Evaluation Scores Algorithm, and the generated code was scored to ensure that the code quality was feasible and highly operable. The experimental results prove the effectiveness of the proposed methods.

Acknowledgment

CLAIM: This work was supported in part by the National Key Research and Development Program of China (2024YFB4709100), the National Nature Science Foundation of China (61773368), Nature Science Foundation of Liaoning province under(2024-MSBA-82), the Research Program of the Liaoning Liaohe Laboratory(No. LLL23ZZ-04-01), and Liaoning Provincial Science and Technology Plan Project (2023JH26/10100004).

References

[1] W. Ling, P. Blunsom, E. Grefenstette, K. Hermann, T. Kočiský, and F. Wang, "Latent Predictor Networks for Code Generation," in Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, 2016, pp. 599-609. doi: 10.18653/v1/P16-1057.

[2] P. C. Yin and G. Neubig, "A Syntactic Neural Model for General-Purpose Code Generation," in Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, 2017, pp. 440-450. doi: 10.18653/v1/P17-1041.

[3] Z. Y. Sun, Q. H. Zhu, Y. F. Xiong, Y. C. Sun, L. L. Mou, and L. Zhang, "TreeGen: A Tree-Based Transformer Architecture for Code Generation," in Proceedings of the AAAI Conference on Artificial Intelligence, 2020, vol. 34, pp. 8984-8991. doi: 10.1609/aaai.v34i05.6430.

[4] B. L. Wei, G. Li, X. Xia, Z. Y. Fu, and Z. Jin, "Code Generation as a Dual Task of Code Summarization," in Proceedings of the 33rd International Conference on Neural Information Processing Systems, 2019, pp. 6563-6573.

[5] A. Kanade, P. Maniatis, G. Balakrishnan, and K. S. Shi, "Learning and Evaluating Contextual Embedding of Source Code," arXiv preprint arXiv:2001.00059, 2020. Available: https://arxiv.org/abs/2001.00059.

[6] Z. Y. Feng, D. Y. Guo, D. Tang, N. Duan, X. C. Feng, M. Gong, L. J. Shou, B. Qin, T. Liu, D. X. Jiang, and M. Zhou, "CodeBERT: A Pre-trained Model for Programming and Natural Languages," in Proceedings of the 2020 Findings of the Association for Computational Linguistics: EMNLP 2020, 2020, pp. 1536-1547. doi: 10.18653/v1/2020.findings-emnlp.139.

[7] D. Y. Guo, S. Ren, S. Lu, Z. Y. Feng, D. Tang, S. J. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, and S. Y. Fu, "GraphCodeBERT: Pre-training Code Representations with Data Flow," arXiv preprint arXiv:2009.08366, 2021. Available: https://arxiv.org/abs/2009.08366.

[8] OpenAI, "GPT-3," 2020. Available: https://openai.com/blog/gpt-3.

[9] GitHub, "Copilot," 2023. Available: https://github.com/features/copilot/.

[10] OpenAI, "GPT-4," 2023. Available: https://openai.com/research/gpt-4.

[11] A. Mejia, A. F. Guarnizo, and G. Barbieri, "Assessment of the PLC Code generated with the GEMMA-GRAFCET Methodology," in Procedia Computer Science, vol. 200, pp. 2192-2200, 2022. doi: 10.1016/j.procs.2022.01.268.

[12] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT," arXiv preprint arXiv:2302.11382, 2023. Available: https://arxiv.org/abs/2302.11382.

[13] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt, "ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design," arXiv preprint arXiv:2303.07839, 2023. Available: https://arxiv.org/abs/2303.07839.

[14] E. Nijkamp, B. Pang, H. Hayashi, L. F. Tu, H. Wang, Y. Zhou, S. Savarese, and C. M. Xiong, "CodeGEN: An Open Large Language Model for Code with Multi-Turn Program Synthesis," arXiv preprint arXiv:2203.13474, 2023. Available: https://arxiv.org/abs/2203.13474.

PLC-100 Dataset and Experiment :
https://github.com/ZzzJjt/PLC-100