PLC Programming Requirement Parsing Method Based on Sequential Function Chart

Zhijun Feng, Yiyang Liu, Lejun Fu, Hao Cui, Xiaoqing Tang, Feng Yu, Yukai Fu, *Member, IEEE*

***Abstract*—In the field of industrial automation, the ambiguity of unstructured PLC programming requirements often leads to symbolic hallucinations and difficulties in precise logical semantic parsing. To address these issues, this paper proposes a requirement parsing method based on Sequential Function Chart (SFC) formal constraints. Firstly, by introducing a symbol anchoring strategy and SFC logical constraint modeling mechanism, the proposed method accurately transforms unstructured natural language requirements into SFC state machine models compliant with the IEC 61131-3 standard. This process aims to establish a formal mapping between natural language and industrial control logic, fundamentally enhancing the rigor of logical expression. Secondly, the parsing method is integrated into a closed-loop framework for PLC program generation and verification, covering the entire workflow from requirement analysis to executable program generation. Finally, experimental results on a dataset of 63 industrial sequential control tasks demonstrate that this method significantly improves the syntactic pass rate and logical accuracy of PLC programs, outperforming the baseline metrics of general Large Language Models (LLMs).**

Keywords—Large Language Models (LLMs), Sequential Function Chart (SFC), Requirement Parsing, Program Verification

I. INTRODUCTION

In recent years, with the continuous deepening of global industrial intelligence strategies and the comprehensive advancement of smart manufacturing technologies, automated modeling and programming of industrial control systems have gained extensive attention in the engineering field [1-2]. As the "industrial brain" at the core of manufacturing systems, the Programmable Logic Controller (PLC) determines the operational stability and productivity of production lines directly through the efficiency and quality of its control logic development [3]. However, the traditional PLC program development lifecycle—covering requirements analysis, code writing, and offline debugging—has long been heavily dependent on manual experience. This development paradigm is not only time-consuming and inefficient but also, due to the lack of unified standardized constraints, prone to inconsistent interface definitions and redundant logic designs. More critically, a significant semantic gap exists between unstructured requirements described in natural language and PLC programs that adhere to strict syntactic rules.

In application scenarios such as precision manufacturing and complex chemical processes, control logic often involves extremely intricate sequential interlocking and state transition relationships [4]. The traditional development model, which relies excessively on manual verification and on-site commissioning, has made the iteration speed of PLC program development unable to keep pace with the growing demand for flexibility and high reliability in modern industrial systems [5]. Therefore, exploring new paradigms for automated PLC program generation based on artificial intelligence technologies such as Large Language Models (LLMs), and solving the mapping challenge from unstructured requirements to industrial standard code, has become a critical engineering problem that urgently needs to be addressed in the current field of industrial control [6-7].

To address the above limitations, this paper first proposes a PLC programming requirements parsing method based on formal constraints of Sequential Function Chart (SFC). This method employs SFC as the core logical bridge linking ambiguous natural language to precise control code, innovatively introducing a symbol anchoring strategy and an SFC modeling completeness assessment mechanism. Through multi-dimensional logical constraints, the method accurately reconstructs fuzzy, unstructured natural language requirements into SFC state machine models compliant with the IEC 61131-3 standard. This process achieves a mathematical mapping from a discrete semantic space to a formal logic space, which not only fundamentally suppresses the inherent hallucination problems of LLMs in variable naming and sequential logic but also provides a formal foundation characterized by mathematical completeness and logical rigor for subsequent program generation, thereby significantly enhancing the standardization and interpretability of requirement expression.

Second, to validate the rationality and engineering practicality of the proposed parsing method, this paper integrates the method into a constructed collaborative framework for PLC program generation and verification, achieving closed-loop automation of the entire PLC development process. The framework takes the standardized SFC modeling results as the core driving input and incorporates a semantic feedback iterative repair mechanism based on the MATIEC compiler, as well as a functional automated verification module relying on an

* Research supported by the following foundations: National Science and Technology Major Project (No. 2025ZD1607400), National Key Research and Development Program of China (No. 2024YFB4711302), Fundamental Research Project of SIA (No. 2024JC1K13), Natural Science Foundation of Jiangsu Province (No. BK20251712), National Natural Science Foundation of China (No. 52541501).

F. Zhijun Feng is with Shenyang University of Chemical Technology, Shenyang 110000, China. (Email: 3387181390@qq.com).

S. Yiyang Liu is with the Institute of Industrial Artificial Intelligence, Chinese Academy of Sciences, Nanjing 211100, China. (Email: yyliu@iaii.ac.cn).

T. Lejun Fu is with the Institute of Industrial Artificial Intelligence, Chinese Academy of Sciences, Nanjing 211100, China. (Email: ljfu@iaii.ac.cn).

F. Hao Cui is with Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China. (Email: cuihao1@sia.cn).

F. Xiaoqing Tang is with the Institute of Industrial Artificial Intelligence, Chinese Academy of Sciences, Nanjing 211100, China. (Email: xiaoqt@iaii.ac.cn).

S. Feng Yu is with Shenyang University of Chemical Technology, Shenyang 110000, China. (Email: lnasyufeng@126.com).

S. Yukai Fu is with the Institute of Industrial Artificial Intelligence, Chinese Academy of Sciences, Nanjing 211100, China. (Corresponding author. Email: fuyukai@sia.cn).

Integrated Development Environment (IDE) [17-18]. Based on comprehensive experiments and ablation comparisons conducted on 63 industrial scenario tasks, this paper verifies the significant advantages of the proposed method in improving program syntactic pass rates and functional accuracy. The results demonstrate that integrating formal requirements parsing into the generation framework is a critical pathway for addressing long-sequence logic control tasks and enhancing the engineering usability of generated code.

II. Procedure for Paper Submission

To address the above challenges, extensive research has been conducted by the academic community both domestically and internationally on large language model-assisted PLC code generation, focusing primarily on two major directions: generation performance enhancement and prompt optimization.

A. Large Model Generation Enhancement

Koziolek et al. deeply explored the potential of general-purpose large models such as GPT-4 in industrial control logic generation. By constructing a multi-category prompt set, they validated the feasibility of large language models generating code compliant with the IEC 61131-3 standard. The team subsequently further introduced Retrieval-Augmented Generation (RAG) technology, which, by integrating a domain-specific knowledge base, significantly reduced the problem of domain knowledge hallucination in the generated code [8-9]. Tran et al. evaluated the performance of various general-purpose large models on PLC programming tasks, designed a reusable evaluation workflow, and quantitatively analyzed the syntactic correctness of each model when processing standard control logic [10]. Furthermore, addressing the issue of semantic ambiguity in natural language requirements, Boudribila et al. constructed the AutoFactory dataset containing a large number of normative texts and, based on a fine-tuned BERT model, achieved efficient extraction of key control components and logic generation [11].

B. Multi-Agent Collaborative Framework

In addition to direct code generation, another line of research focuses on program verification architectures and multi-agent collaborative mechanisms, aiming to enhance the engineering practicality and system adaptability of generated code. The LLM4PLC framework proposed by Fakih et al. emphasizes the post-generation verification stage, employing fine-tuned models to perform rigorous syntax checking and error correction, ensuring that the code complies with industrial safety standards [12]. The LLM4VC framework introduced by Lyu et al. applies large language models to the field of virtual commissioning; through automated generation of simulation models, it substantially reduces the labor cost of test environment construction [13]. At the system-level collaboration level, the Agents4PLC framework constructed by Liu et al. establishes a closed-loop workflow of code generation and verification through multi-role agent collaboration [14]. Meanwhile, Ren et al. and Yang et al., respectively through fine-tuning on large-scale datasets and constructing vendor-specific knowledge bases, solved the heterogeneous adaptation challenge among different PLC hardware brands[15], effectively supporting the automatic generation of cross-platform Structured Text (ST) code [16].

Although existing research has achieved significant progress in improving the syntactic correctness of code generation and multi-modal collaboration, current end-to-end generation paradigms still suffer from inherent limitations when processing long-sequence industrial logic. Due to the lack of interpretable intermediate logical representation, large language models struggle to maintain long-range logical consistency during the direct translation of natural language into low-level code, making them prone to logical discontinuities or state loss. This black-box generation mode implies that even if the generated code is syntactically flawless, its internal control timing logic may deviate considerably from actual process requirements. Moreover, current verification mechanisms mostly focus on back-end static syntax error correction and simulation-based validation, neglecting the standardized parsing of requirements at the front-end input stage. If the model's understanding of the input requirements is inherently ambiguous, all subsequent code optimization will inevitably be built upon a flawed logical foundation.

III. Methodology and System Framework

A. Requirement Parsing and Modeling Based on SFC Formal Constraints

To establish a precise mapping between natural language requirements and industrial control logic, this study constructs a mapping mechanism from the unstructured requirement space R_{user} to the formal logic space S_{SFC}. In contrast to the black-box paradigm of direct generation, this mechanism introduces the SFC as a physically meaningful intermediate representation, transforming ambiguous semantics into a rigorous state machine model through mathematical mapping. This paper first defines the formal logic space as a complete triplet set $S_{SFC} = \{S, T, A\}$, where S, T, and A represent finite $S_{SFC} = \{S, T, A\}$ sets of state steps, transition conditions, and actions, respectively. The parsing process executes the mapping function F_{dec}. Under the constraints of the predefined symbol table F_{sym}, it utilizes the state extraction operator M_{step}, the transition construction operator M_{trans}, and the action mapping operator M_{act} to perform "event-response" decoupling on the unstructured descriptions in R_{user} and maps them to the aforementioned three sets, respectively. This mapping relationship can be specifically expressed as Eq. (1)

$$\begin{cases} S \leftarrow M_{step}(R_{user}) \\ T \leftarrow M_{trans}(R_{user}, F_{sym}) \\ A \leftarrow M_{act}(R_{user}, F_{sym}) \end{cases} \tag{1}$$

Specifically, M_{step} is responsible for extracting stage descriptions with temporal pause characteristics from the text to construct the step set S of the SFC, such as explicit descriptions including "initial state" and "grasping stage". M_{trans} aims to identify logical predicates or physical signals that trigger state transitions, such as state transition specifications like the start switch being true and the in-place detection signal being valid, so as to define the transition condition set T. M_{act} establishes the correlation between steps and specific execution logic, and accurately assigns instructional descriptions in the requirements to the action set A, such as "opening the valve" and "electric cylinder extending". This decoupled mapping mechanism ensures that the vague logic contained in natural language can

be precisely decomposed into functional units that comply with industrial standards.

Upon completion of discrete element extraction, the algorithm performs a multi-dimensional topological reconstruction of the task state machine based on graph theory. The reconstructed SFC model is represented as a directed bipartite graph $G = (V, E)$, where the node set is defined as $V = S \cup T$. To ensure the generated control flow complies with the IEC 61131-3 standard, an alternating connection operator Ω is introduced to strictly constrain the construction of the edge set E. From a mathematical structural perspective, this operator rigorously limits the connection paradigm of edges, precluding illicit structures such as "step-to-step" or "transition-to-transition" connections, thereby eliminating the structural risk of control deadlocks. Furthermore, the reconstruction process involves a deep fusion of data and temporal dimensions: data dimension reconstruction establishes the binding relationship between state nodes and action blocks, while temporal dimension reconstruction is responsible for identifying cyclic logic by constructing feedback edges within the graph to explicitly form strongly connected components, thereby recovering complex industrial cyclic control intentions. The edge construction rules are described in Eq. (2):

$$E = \Omega(S, T) \subseteq (S \times T) \cup (T \times S) \quad (2)$$

B. Symbol Anchoring Strategy and Modeling Completeness Assessment

To address the common issue of "variable hallucinations" inherent in Large Language Models during code generation, this study implements a mandatory symbol anchoring strategy. This strategy first defines a standard symbol space $F = \{v_i | v_i = (n_i, \tau_i, a_i, d_i)\}_{i=1}^{N}$, where each element comprises a normative variable identifier, data type, physical address, and natural language description. The symbol anchoring process is modeled as a multi-objective retrieval optimization problem: for each semantic entity e extracted from requirements, the system anchors it to the unique optimal variable v^* in the symbol space by calculating the weighted sum of semantic similarity $S(\cdot)$ and type constraint $C(\cdot)$. Fig. 1 illustrates the schematic diagram of this semantic matching principle, demonstrating how the system filters candidate variables based on type constraints and identifies the optimal match via semantic similarity.

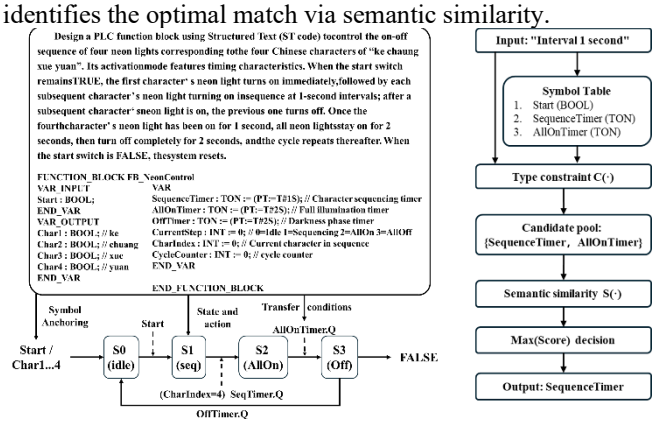

Fig. 1 Workflow of the symbol anchoring strategy based on semantic similarity and type constraints.

This dual-constraint mechanism ensures that the generated SFC model is strictly consistent with the hardware configuration table at the symbolic level, thereby preventing the introduction of non-standard variables at the source. The calculation formula is shown in Eq. (3):

$$v^* = \text{argmax}_{v_i \in F} \left[\alpha \cdot S(e, d_i) + \beta \cdot C(e, \tau_i)\right] \quad (3)$$

Among them, α and β denote the weight coefficients of semantic similarity and type constraint, respectively. α reflects the degree to which the system adheres to the consistency of natural language descriptions, while β determines the mandatory level of type matching. By adjusting the ratio of α to β, the anchoring results can achieve targeted requirement parsing under different unstructured demands. A larger α helps infer the target variable from context when variable naming is non-standard, whereas a larger β can effectively eliminate invalid matches with similar names but conflicting data types. For instance, it avoids incorrectly mapping Boolean trigger signals to floating-point sensor values, thereby ensuring that the generated SFC model is strictly consistent with the hardware configuration table at the symbolic level.

To prevent the propagation of logical defects to the downstream code generation stage, this paper further constructs a quality assessment standard based on an evaluation function *J(G)*. The quality score ranges from 0 to 30 points, quantitatively evaluating the generated SFC model G across three core dimensions: structural integrity (S_{str}), symbol consistency (S_{sym}), and constraint coverage (S_{cov}), with each dimension assigned a maximum value of 10 points. Specifically, structural integrity detects whether the topological structure adheres to the "step-transition" alternation rule ; symbol consistency evaluates the accuracy of variable anchoring; and constraint coverage measures the completeness of mapping user explicit requirements within the model.

$$J(G) = S_{str} + S_{sym} + S_{cov} \quad (4)$$

Fig. 2 presents the schematic workflow of this modeling completeness assessment, detailing the calculation logic for the three evaluation metrics and the decision pathway for triggering the feedback optimization loop.

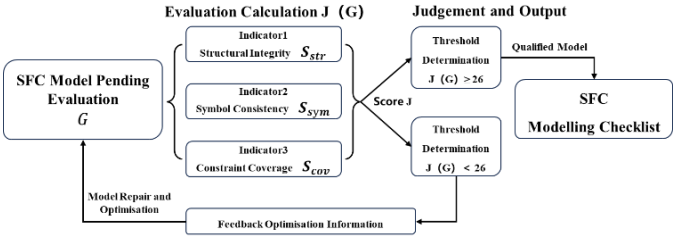

Fig. 2 Schematic diagram of the modeling completeness assessment based on the evaluation function *J*

This study sets a quality threshold $J_{th} = 26$. This value was determined through empirical manual inspection and cross-validation of modeling manifests across 63 industrial scenario tasks. Statistical results from the experimental phase indicate that when the cumulative deduction exceeds 4 points (i.e., *J(G)* < 26), the modeling manifests typically harbor fatal defects such as missing critical state transitions or severe variable type deviations, which were found insufficient to support the model in completing program generation tasks during testing. Consequently, if and only if *J(G)* $\geqslant$ 26, the model is permitted to proceed to the code generation stage ; otherwise, the system classifies the model as "logically distorted" and transmits specific defect feedback information back to the requirement parsing module to trigger logical correction, thereby forming a

"generation-evaluation-optimization" closed-loop mechanism. The quality scoring formula is defined as Eq. (4):

C. Closed-loop PLC Program Generation and Verification Framework

Based on the aforementioned formal requirement parsing method, this paper constructs a modular collaborative framework spanning the entire lifecycle of "requirement parsing - program generation - syntax verification - functional verification," as illustrated in Fig. 3. The core innovation of this framework lies in reconstructing the traditional "unidirectional generation" mode into a "generation-verification-correction" cyclic evolutionary paradigm. Specifically, the program generation module utilizes the standardized SFC modeling manifest as the logical foundation to drive the LLM to generate ST code compliant with the IEC 61131-3 standard. Addressing potential syntax defects in the initial code, the framework integrates the MATIEC compiler as the core verification tool. Upon detecting compilation errors, the system automatically parses the standardized error stream output by the compiler, precisely extracting the line number of the error occurrence, character offset, and specific semantic feature descriptions.

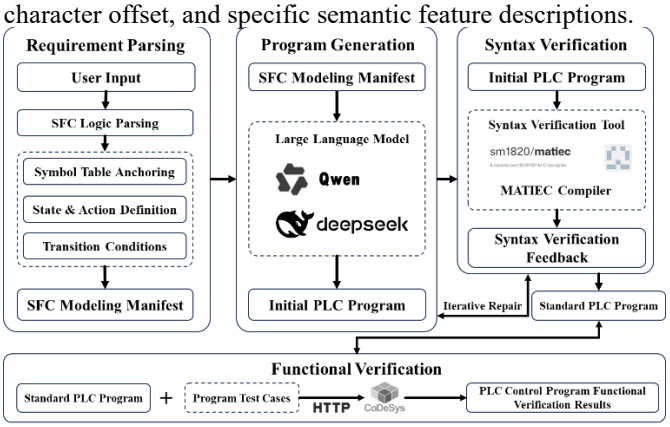

Fig. 3 The overall architecture of the closed-loop PLC program generation and verification framework.

Subsequently, this strategy abstracts the repair process as a constrained conditional generation optimization problem. By fusing the extracted error features with the source code fragments, it constructs strongly guided structured repair instructions, denoted as I_{fix}, which are fed back to the generation model. This guides the model to perform targeted closed-loop iterative repairs until the code passes all static syntax checks.

On the basis of ensuring syntactic correctness, the framework interacts with CODESYS through HTTP interfaces to realize automatic functional verification of PLC code. To guarantee the accuracy of code verification, the function verification module constructs a standard test program equipped with a BOOL judgment interface, and integrates it with the program under test into a standard test project. This test program calculates the execution results of all functional test cases via logical AND operation, which serves as the only output indicator of functional correctness. If and only if all test cases pass the verification, the system determines that the generated PLC code fully meets user requirements, thereby achieving end-to-end automated closed-loop verification of PLC code.

IV. Experimental Results and Discussion

Given the absence of specialized test datasets in existing research tailored for PLC program generation involving sequential control logic, this study constructed a dataset covering 63 common PLC programming tasks in industrial scenarios to facilitate the experimental verification. This dataset comprises user requirements, control program interface definitions, ground truth reference programs, and functional test cases.

To quantitatively evaluate the performance, experiments are conducted to analyze the comparative performance of different strategies based on the following metrics:

- Syntax Pass Rate (R_{syn}): The proportion of code that passes the MATIEC compiler verification. The calculation method refers to Eq. (5):

$$R_{syn} = \frac{N_{pass}}{N_{total}} \tag{5}$$

  where N_{pass} represents the number of tasks that passed the compiler verification without errors, and N_{total} is the total number of tasks in the dataset, with $N_{total} = 63$.
- Functional Accuracy (R_{acc}): The proportion of code that satisfies all functional requirements after simulation verification and passing all test cases. The calculation method refers to Eq. (6):

$$R_{acc} = \frac{N_{logic}}{N_{total}} \tag{6}$$

  where N_{logic} represents the total number of tasks that fully satisfy the requirements of all corresponding test cases in the simulation environment.

The experimental group adopting SFC modeling must satisfy the modeling evaluation criteria before starting the code generation phase. If the evaluation score is lower than the preset threshold J_{th}, the requirement parsing and modeling process will be re-executed iteratively until the evaluation criteria are met. This experiment conducts a comparative analysis of four generation strategies, namely Base-LLM, Base-Close, SFC-Open, and SFC-LLM. The specific definition of each strategy is given as follows:

- Base-LLM: This baseline strategy utilizes the unstructured natural language user requirements directly from the test dataset as the raw input to drive the LLM for code generation. Crucially, the code generated under this strategy does not undergo any subsequent syntax iterative repair processes.
- Base-Close: This strategy introduces a syntax iterative repair mechanism on the basis of code generation by the base model. It takes unstructured programming requirements as model input to guide the large language model to generate code. The code generated in this group undergoes syntax iterative repair, and the repaired code is then evaluated.
- SFC-Open: This strategy employs the proposed SFC logical constraint modeling method to parse and reconstruct the unstructured user requirements. The resulting standardized modeling manifest, which has passed the quality assessment, serves as the structured input to guide the LLM in generating code. Similar to the Base-LLM strategy, the generated code in this group is evaluated without applying any syntax iterative repair.

- SFC-Close: This strategy integrates the proposed SFC logical constraint modeling method with an iterative repair mechanism to parse and reconstruct unstructured user requirements, and performs iterative syntax repair on the generated programs. The standardized modeling results that pass quality assessment are taken as structured input to guide the large language model in code generation. The code generated in this group undergoes the syntax iterative repair process, and the repaired programs are subsequently evaluated.

The statistical results of the syntax pass rate and logical accuracy for each experimental group are shown in Table 1.

TABLE Ⅰ GENERAL EXPERIMENTAL RESULTS USING LLMS OF VARYING TYPES AND PARAMETER SIZES

R_{syn} denotes the syntax accuracy rate of the generated code, while R_{acc} represents the functional accuracy rate of the code.

Base model	**General Experimental Results**		
	Comparison category	R_{syn}	R_{acc}
Deepseek-r1	Base-LLM	0.873	0.413
	Base-Close	0.952	0.413
	SFC- Open	0.889	0.429
	SFC-Close	0.984	0.429
Deepseek-v3	Base-LLM	0.889	0.317
	Base-Close	0.952	0.317
	SFC- Open	0.889	0.349
	SFC-Close	0.968	0.365
Qwen-max	Base-LLM	0.873	0.365
	Base-Close	0.952	0.381
	SFC- Open	0.873	0.394
	SFC-Close	0.984	0.413
Qwen2.5-72b	Base-LLM	0.794	0.143
	Base-Close	0.952	0.143
	SFC- Open	0.873	0.159
	SFC-Close	0.968	0.190
Qwen2.5-coder-32b	Base-LLM	0.762	0.190
	Base-Close	0.905	0.206
	SFC- Open	0.778	0.238
	SFC-Close	0.905	0.238

The results in Table I demonstrate that the introduction of SFC-based logic constraint modeling and a closed-loop repair mechanism significantly improves PLC code generation performance for all tested models, substantially increasing both the syntax accuracy rate (R_{syn}) and the functional accuracy rate (R_{acc}).

A comparison from the Base-LLM setting to the Base-Close setting reveals that pure iterative syntax repair can effectively raise R_{syn}. For instance, the R_{syn} of Deepseek-r1 increases from 0.873 to 0.952, and that of Qwen2.5-72b rises from 0.794 to 0.952. However, the improvement in R_{acc} for the Base-Close group remains marginal; the R_{acc} of Deepseek-r1 stays at 0.413, confirming that syntax repair alone can hardly fix functional or logical defects.

A comparison from the Base-LLM setting to the SFC-Open setting validates the core role of SFC logic modeling. Even without introducing syntax repair, merely converting unstructured requirements into an SFC modeling list yields an increased R_{acc} for all models. Specifically, Deepseek-v3 increases from 0.317 to 0.349, and Qwen-max from 0.365 to 0.394. This indicates that SFC, as an intermediate representation with clear physical significance, effectively suppresses logical hallucinations induced by long-text requirements through the decoupling of states, transitions, and actions.

A comparison from the SFC-Open setting to the SFC-Close setting shows that integrating front-end SFC modeling with back-end iterative repair achieves the best performance. Both Deepseek-r1 and Qwen-max reach an R_{syn} of 98.4%, with their R_{acc} also peaking among all groups. Other models also show notable gains. The front-end symbol anchoring strategy reduces variable-name hallucinations at the source, and the back-end syntax repair guarantees the engineering usability of the generated code.

In terms of model characteristics, reasoning-oriented large models achieve high code generation accuracy under the baseline LLM setting by virtue of their powerful reasoning capabilities. However, specialized code generation models perform slightly worse than general-purpose large models in PLC code generation tasks. The underlying reason is that these models are trained on general programming languages such as Python and C, which introduces interference in PLC code generation. Guided by the SFC model, general-purpose large models can enhance their logical reasoning ability and handle industrial control tasks of simple and medium complexity without domain-specific fine-tuning.

The ablation results presented in Table 2 are used to verify the independent contributions of the symbol anchoring strategy and the modeling evaluation mechanism in the SFC parsing and modeling process.

TABLE Ⅱ ABLATION STUDY RESULTS: IMPACT ANALYSIS OF KEY COMPONENTS ON MODEL PERFORMANCE

In the table below, R_{syn} denotes the syntax compilation pass rate of the generated code, and R_{acc} represents the functional accuracy of the program.

Base model	**General Experimental Results**		
	Comparison category	R_{syn}	R_{acc}
Deepseek-r1	SFC-Close	0.984	0.429
	w/o Anchoring	0.952	0.397
	w/o J-Auditing	0.968	0.397
Deepseek-v3	SFC-Close	0.968	0.365
	w/o Anchoring	0.857	0.349
	w/o J-Auditing	0.889	0.333
Qwen-max	SFC-Close	0.984	0.413
	w/o Anchoring	0.857	0.365
	w/o J-Auditing	0.873	0.333
Qwen2.5-72b	SFC-Close	0.968	0.190

Base model	General Experimental Results		
	Comparison category	R_{syn}	R_{acc}
	w/o Anchoring	0.794	0.190
	w/o J-Auditing	0.809	0.159
Qwen2.5-coder-32b	SFC-Close	0.905	0.238
	w/o Anchoring	0.777	0.206
	w/o J-Auditing	0.873	0.191

The ablation results presented in Table II profoundly underscore the core value and impact of the Symbol Anchoring strategy and the Modeling Assessment (J-Auditing) mechanism in ensuring the syntactic and logical accuracy of PLC code. Upon the removal of the Symbol Anchoring module (w/o Anchoring), the syntax pass rate R_{syn} across all models exhibited a significant decline. For instance, the R_{syn} of Qwen2.5-72b plummeted from 96.8% to 79.4%, and Deepseek-v3 also experienced a drop of over 11 percentage points. This compellingly demonstrates that symbol anchoring is not merely a simple naming match, but a mandatory constraint at the physical level.

By strictly confining the generation space within a predefined symbol table, it fundamentally intercepts variable definition hallucinations resulting from the free divergence of LLMs, thereby constituting the first line of defense for code compilability. Conversely, removing the J-value threshold constraint (w/o J-Auditing), while having a minimal impact on the syntax pass rate, resulted in a universal regression in functional accuracy (R_{acc}); for example, the functional accuracy of Deepseek-r1 declined from 42.9% to 39.7%. This phenomenon indicates that the J-value effectively guarantees the quality of the logical parsing model, preventing inferior modeling manifests—characterized by chaotic logic or sequential defects—from entering the generation phase. In the absence of this admission criterion, even if the generated code is syntactically correct, its intrinsic control logic often harbors fatal defects such as deadlocks or state losses. This confirms that a standardized SFC modeling manifest is a prerequisite for ensuring the logical completeness of the generated code.

V. CONCLUSION

To address the logical parsing challenges in the automatic generation of PLC programs, this paper proposes an SFC-based requirement parsing method with formal constraints and a closed-loop collaborative framework. By mapping ambiguous natural language requirements into complete SFC models, precise parsing of programming requirements is realized. The proposed framework can effectively mitigate variable hallucinations and logical deviations in sequential control. Experimental results show that with the adopted strategy, the syntax pass rate of the generated code exceeds 90%, and the functional accuracy is significantly improved compared with baseline models. This further verifies the effectiveness of the technical pathway in enhancing the engineering application reliability of PLC code.

Future work will focus on constructing high-quality IEC 61131-3 datasets for Supervised Fine-Tuning (SFT) and exploring generation paradigms based on Multi-Agent collaboration to further enhance the intrinsic capabilities and adaptability for complex industrial scenarios.

REFERENCES

[1] Zhang Tao, Yang Bingheng, Wang Yankai. PID Control of Stepper Motor Based on S7-1500 PLC [J]. Computer and Digital Engineering, 2024, 52(02):635-640.

[2] Zhao Jia. Application of AI Large Models in the Teaching of Higher Vocational Course "Electrical and PLC Control Technology" [J]. China Electric Power Educa-tion,2025,(04):72-73.

[3] Zhang Tao, Li Lin, Lei Chao. Research on PLC Programming Application Based on Artificial Intelligence[J]. Industrial Control Com-puter, 2024, 37(11):128-130.

[4] Ma Daobin, Ren Jingjun, Liu Xuan, et al. PLC Structured Text Code Generation Based on Functional Element Guidance and Multi-Agent Collaboration [J/OL]. Journal of Beijing University of Aeronautics and Astronautics, 1-15 [2026-01-14].

[5] Xu Ao. Research and Implementation of Code-Generating AI Models for Intelligent Manufacturing [D]. Shenyang University of Technology, 2025. DOI:10.27322/d.cnki.gsgyu.2025.000882.

[6] Li Hongyu, Yuan Shen. Application of Structured Text Programming in Motion Control Systems[J]. Heilongjiang Metallurgy, 2015, 35(01):36-38.

[7] K. Tran, J. Zhang, J. Pfeiffer, A. Wortmann and B. Wiesmayr, "Generating PLC Code with Universal Large Language Models," 2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA), Padova, Italy, 2024, pp. 1-8.

[8] H. Koziolek, S. Gruener and V. Ashiwal, "ChatGPT for PLC/DCS Control Logic Generation," 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), Sinaia, Romania, 2023, pp. 1-8.

[9] H. Koziolek and A. Koziolek, "LLM-based Control Code Generation using Image Recognition," 2024 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code), Lisbon, Portugal, 2024, pp. 38-45.

[10] Boudribila, A., Tajer, A., Boulghasoul, Z. (2025). Automatic generation of PLC control code from natural language requirement specifications. Ingénierie des Systèmes d'Information, Vol. 30, No. 6, pp. 1597-1607.

[11] Pei, Pengcheng. "Multi-Agent System for Cross-Platform PLC Code Generation with Domain Adaptation." International Journal of Emerging Technologies and Advanced Appli-cations 2.10 (2025): 1-8.

[12] M. Fakih, R. Dharmaji, Y. Moghaddas, G. Q. Araya, O. Ogundare and M. A. Al Faruque, "LLM4PlC: Harnessing large Language Models for Verifiable Programming of PlCs in In-dustrial Control Systems," 2024 IEEE/ACM 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Lisbon, Portugal, 2024, pp. 192-203.

[13] T. Lyu, U. D. Atmojo and V. Vyatkin, "LLM4VC: Harnessing Large Language Models for Virtual Commissioning of IEC 61499 Automation Systems," 2025 IEEE 34th International Symposium on Industrial Electronics (ISIE), Toronto, ON, Canada, 2025, pp. 1-6.

[14] Z. Liu et al., "Agents4PLC: Automating Closed-loop PLC Code Generation and Verification in Industrial Control Systems using LLM-based Agents," in IEEE Transactions on Software Engineering,2024: 1-12.

[15] Ren, L., Wang, H., Dong, J., Wang, H., Liu, S., Laili, Y., & Zhang, L. (2025). Me-taIndux-PLC: Un LLM guiado por lógica de control para la generación de código PLC en sistemas de control industrial. Applied Soft Computing, 184.

[16] D. Yang et al., "AutoPLC: Generating Vendor-Aware Structured Text for Programmable Logic Controllers," 2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE), Seoul, Korea, Republic of, 2025, pp. 3593-3604.

[17] M. de Sousa and nucleron, "matiec - IEC 61131-3 compiler," GitHub, 2017. [Online]. Available: https://github.com/nucleron/matiec, Jan. 14, 2026.

[18] Roberto Cavada "The nuXmv symbolic model checker ". In: Computer Aided Verification: 26th International Confer-ence, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18–22, 2014. Proceedings 26. Springer. 2014, pp. 334–342