International Journal of Emerging Technologies and Advanced Applications

# Multi-Agent System for Cross-Platform PLC Code Generation with Domain Adaptation

Pengcheng Pei[1]
[1]Liaoning Institute of Science and Technology, China
349692726@qq.com

***Abstract*—Programmable Logic Controllers (PLCs) are fundamental to industrial automation systems. However, traditional PLC programming requires extensive domain expertise and significant time investment, while code reusability remains limited and cross-platform adaptation poses substantial challenges. With the rapid advancement of Large Language Models (LLMs), LLM-based code generation offers a promising approach to address these issues. Nevertheless, existing methods still face challenges when handling complex industrial scenarios, including insufficient domain knowledge, unstable code quality, and weak cross-platform adaptation capabilities. This paper presents a multi-agent system for intelligent cross-platform PLC code generation, featuring a collaborative framework consisting of four specialized agents: requirement analysis, architecture design, code generation, and verification-optimization. The method injects domain knowledge through a Retrieval-Augmented Generation (RAG) mechanism, employs multi-stage prompt engineering strategies to guide code generation, and integrates a three-layer verification mechanism comprising static analysis, dynamic simulation, and expert review to ensure code quality. Experiments on the constructed PLC-MultiTask dataset demonstrate that our method significantly outperforms existing approaches across multiple metrics, achieving 90.3% compilation success rate, 87.6% test pass rate, and 75.4 CodeBLEU score. In an industrial case study involving robotic arm handling of refractory bricks, the system successfully generated approximately 800 lines of structured text code. Field testing over 720 hours demonstrated stable operation with 99.2% handling success rate, reducing development time by 73.3% compared to traditional methods. These results indicate that multi-agent-based PLC code generation significantly enhances development efficiency, ensures code quality, and strengthens cross-platform adaptation capabilities, offering a novel paradigm for industrial automation software development.**

***Index Terms*—PLC code generation, Multi-agent systems, Large language models, Retrieval-augmented generation, Industrial automation**

## I. Introduction

As core control devices in industrial automation systems, Programmable Logic Controllers (PLCs) play irreplaceable roles in manufacturing, energy, and chemical engineering sectors. However, with Industry 4.0 and smart manufacturing advancement, modern industrial control systems face unprecedented challenges in PLC programming complexity and reliability [1]. Traditional PLC programming methods heavily depend on domain experts' experience, resulting in lengthy development cycles, high costs, and requiring substantial manual adaptation when facing diverse industrial scenarios and cross-platform compatibility requirements. Particularly in applications involving complex actuators such as robotic arms in high-temperature, high-intensity operations like refractory brick handling, PLC program development must simultaneously address logical control accuracy, safety, real-time performance, and multi-device coordination constraints [2]. This complexity creates severe efficiency bottlenecks and quality concerns in traditional manual programming, substantially restricting rapid deployment and iterative optimization capabilities of industrial automation systems.

In recent years, Large Language Models (LLMs) have achieved breakthrough progress in natural language processing and code generation, providing novel technical pathways for addressing PLC programming challenges. Representative LLMs such as GPT, LLaMA, and Claude demonstrate powerful code comprehension and generation capabilities, enabling automatic generation of high-quality program code from natural language descriptions and making it feasible to transform engineers' requirements into IEC 61131-3 standard-compliant PLC code [3]. However, directly applying general-purpose LLMs to PLC code generation faces numerous challenges: PLC programming languages (such as Structured Text and Ladder Diagram) are Domain-Specific Languages with significant differences from general-purpose programming languages in syntax structure and programming paradigms; industrial control systems impose extremely high reliability and safety requirements necessitating rigorous code verification and testing; and different PLC platforms vary in instruction sets, function libraries, and programming conventions, making cross-platform code generation and adaptation critical challenges requiring resolution.

To address these challenges, this research proposes an intelligent cross-platform PLC code generation method based on multi-agent systems. Drawing on multi-agent collaborative frameworks' advantages in complex task decomposition



ISSN:3006-2985(Print)
ISSN:3006-9297(Online)

Received: Sep 20, 2025
Revised: Oct 23, 2025
Accepted: Oct 25, 2025

and parallel processing, we construct a cooperative system comprising requirement analysis, code generation, verification and testing, and optimization agents, achieving end-to-end automated generation from natural language requirements to executable PLC code. The main contributions include: (1) designing a multi-agent collaborative architecture that enhances code generation accuracy and reliability through inter-agent information exchange and task coordination; (2) proposing a domain knowledge injection mechanism based on Retrieval-Augmented Generation (RAG) that enhances LLMs' understanding of PLC domain knowledge by integrating programming manuals, standard function libraries, and historical code repositories; (3) developing a cross-platform adaptation mechanism that automatically adjusts code generation strategies according to target PLC platform characteristics, achieving unified support for mainstream platforms including Siemens, Rockwell, and Schneider; (4) constructing an evaluation dataset with real industrial scenarios, using robotic arm refractory brick handling as a representative case study to systematically validate the method's effectiveness and practicality in actual industrial environments.

## II. Related Work

### A. Traditional PLC Automatic Programming Methods

Automated programming of PLCs has long been an important research direction in industrial automation. Early research primarily focused on model-based code synthesis methods, which typically require engineers to first establish formalized system models, then generate PLC code through model transformation techniques. Representative modeling languages include formal description methods such as Unified Modeling Language (UML), Petri nets, and temporal logic. While these methods can improve programming standardization and reliability to some extent, they suffer from limitations including high modeling barriers, poor flexibility, and difficulty adapting to complex industrial scenarios. Additionally, rule-based expert systems have been applied to PLC code generation, converting process flow descriptions into control logic through predefined rule bases. However, these approaches heavily depend on the completeness of domain expert knowledge and rule accuracy, often requiring substantial manual adjustment and extension when facing new scenarios [4]. With the development of Industry 4.0, manufacturing systems increasingly demand flexibility and reconfigurability. Traditional automatic programming methods can no longer meet the requirements for rapid iteration and cross-platform deployment, urgently necessitating new technical paradigms to break through this bottleneck.

### B. Applications of Large Language Models in Code Generation

The emergence of LLMs has brought revolutionary changes to automated code generation. Pre-trained models based on Transformer architecture, such as the GPT series, CodeLLaMA, and StarCoder, have acquired powerful code comprehension and generation capabilities through pre-training on massive code corpora. These models can directly generate syntactically correct program code from natural language descriptions, significantly lowering programming barriers [5]. Hou et al. conducted a systematic review of LLM applications in software engineering, indicating that LLMs have achieved significant results in tasks including code completion, vulnerability detection, and program repair [6]. For code generation tasks, researchers have proposed various improvement strategies, including prompt engineering, Retrieval-Augmented Generation (RAG), and Parameter-Efficient Fine-Tuning (PEFT). These methods effectively improve code generation quality and controllability by optimizing input prompts, introducing external knowledge bases, or performing domain adaptation of models. However, general-purpose code LLMs still face challenges when dealing with domain-specific languages, particularly for low-resource programming languages with scarce training data and specialized languages in industrial control domains, where model generation accuracy and reliability often fail to meet practical application requirements [7].

### C. Exploration of Large Language Models in PLC Programming

For the specialized domain of PLC programming, several pioneering research efforts have emerged in recent years. Fakih et al. proposed the LLM4PLC framework, the first work to systematically apply LLMs to PLC code generation. This framework significantly improved PLC code generation success rates from 47% to 72% through a user-guided iterative process combined with syntax checkers, compilers, and SMV formal verification tools [4]. The research also employed Low-Rank Adaptation (LoRA) techniques for domain fine-tuning of open-source models and conducted practical validation on the FischerTechnik manufacturing test platform. Liu et al. further proposed the Agents4PLC framework, implementing closed-loop PLC code generation and verification through an LLM-based multi-agent system. This system includes retrieval agents, planning agents, coding agents, and verification agents, capable of automatically completing the entire process from requirement analysis to code verification [5]. Koziolek et al. explored ChatGPT's application in generating control logic for industrial process control systems (DCS/PLC), validating the feasibility of LLMs understanding process flow diagrams and generating control code [3]. While these research efforts have collectively advanced intelligent PLC programming technology, they also reveal common issues: insufficient cross-platform compatibility support, incomplete domain knowledge injection mechanisms, and verification mechanisms primarily remaining at the design level rather than the code level.

### D. Applications of Multi-Agent Systems in Industrial Automation

As a distributed artificial intelligence technology, multi-agent systems offer broad application prospects in industrial automation. Wang et al. conducted a comprehensive review of autonomous agents based on LLMs, noting that multi-agent collaboration demonstrates unique advantages in com-

plex task decomposition, parallel processing, and collective intelligence emergence [8]. In manufacturing systems, researchers have successfully applied multi-agent technology to scenarios including production scheduling, resource allocation, and quality control. Lim et al. proposed an LLM-enabled multi-agent manufacturing system framework that achieves intelligent collaboration between machines and human-machine interaction through natural language interfaces [9]. This framework enables agents in manufacturing systems to understand human instructions and make autonomous decisions, significantly improving system flexibility and adaptability. In the Industry 4.0 context, Sidorenko et al. investigated automation platform-independent multi-agent systems in production resource networks, demonstrating the effectiveness of multi-agent architectures when addressing complex manufacturing scenarios [10]. These studies inspire us that introducing multi-agent collaboration mechanisms into PLC code generation tasks can better handle the complexity of different stages including requirement understanding, code generation, and verification testing through task decomposition and specialized division of labor. This research, guided by these concepts, combines LLMs' code generation capabilities with multi-agent collaboration mechanisms to construct an intelligent solution for PLC programming.

## III. Methodology

### A. System Architecture

The multi-agent-based cross-platform intelligent PLC code generation system proposed in this research adopts a layered collaborative architecture, aiming to achieve end-to-end automated generation from natural language requirements to executable PLC code. The overall system architecture, illustrated in Figure 1, primarily comprises four core agent modules: Requirement Analysis Agent, Code Generation Agent, Verification Agent, and Optimization Agent. These four agents exchange information and coordinate tasks through Shared Working Memory and a Coordinator, forming a closed-loop feedback mechanism. The system also integrates a Domain Knowledge Base containing PLC programming standards, platform-specific function libraries, historical code repositories, and industrial application case libraries, providing necessary domain knowledge support for all agents. In industrial application scenarios such as robotic arm refractory brick handling, the system must handle complex control logic including precise position control in high-temperature environments, force adjustment, collision detection, and exception handling, imposing extremely high requirements on code generation accuracy and reliability. Through multi-agent collaboration mechanisms, the system can decompose complex control requirements into multiple subtasks handled by specialized agents, thereby improving overall generation quality and efficiency.

### B. Multi-Agent Collaboration Mechanism

The multi-agent collaboration mechanism represents the core innovation of this system. Drawing on collaborative

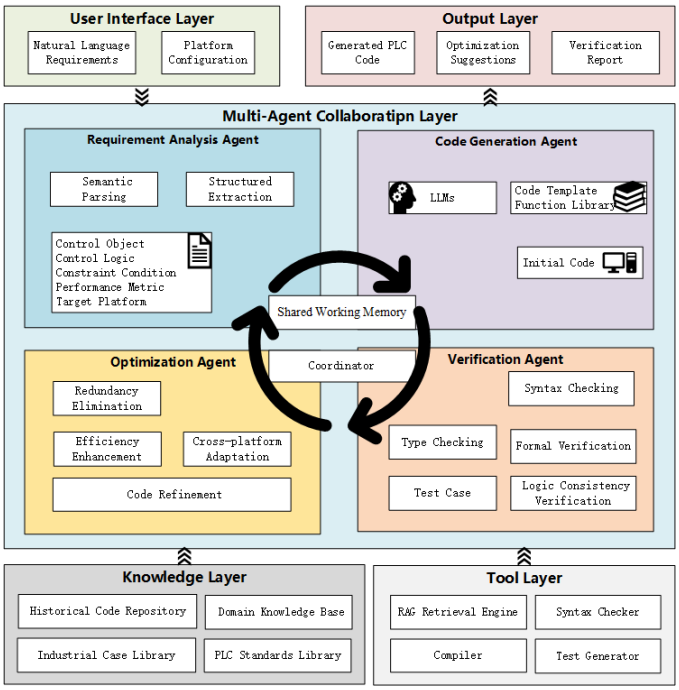


Fig. 1. Architecture of Multi-Agent System for Intelligent PLC Code Generation

strategies from multi-agent systems and adopting the meta-programming collaboration framework concept proposed by MetaGPT [11], we have designed an agent interaction protocol suitable for PLC code generation tasks. The Requirement Analysis Agent first performs semantic parsing and structured extraction of users' natural language input, identifying key elements including control objects, control logic, constraint conditions, and performance indicators. This agent employs Chain-of-Thought reasoning strategies to decompose complex requirements into executable subtask sequences and generate formalized functional specifications. After receiving functional specifications, the Code Generation Agent retrieves relevant code templates and function libraries from the domain knowledge base according to target PLC platform characteristics, utilizing LLMs' code generation capabilities to produce initial code. The Verification Agent conducts multi-level verification of generated code, including syntax checking, type checking, logic consistency verification, and formal verification, while generating test cases for functional testing. Based on verification results and performance indicators, the Optimization Agent improves and optimizes the code, including redundancy elimination, execution efficiency enhancement, and cross-platform adaptation. Inter-agent collaboration follows an improved Contract Net protocol, defining agent roles, responsibilities, and communication specifications to ensure rational task allocation and accurate information transmission.

The collaboration efficiency between agents can be quantitatively evaluated through the collaboration metric function $C$:

$$C = \alpha \cdot \frac{N_{success}}{N_{total}} + \beta \cdot \frac{1}{T_{avg}} + \gamma \cdot \left(1 - \frac{N_{iteration}}{N_{max}}\right) \quad (1)$$

where $N_{success}$ represents the number of successfully completed tasks, $N_{total}$ represents the total number of tasks, $T_{avg}$ represents average task completion time, $N_{iteration}$ represents average iteration count, $N_{max}$ represents maximum allowed iterations, and $\alpha$, $\beta$, $\gamma$ are weight coefficients satisfying $\alpha+\beta+\gamma = 1$. This metric function comprehensively considers three dimensions—task success rate, execution efficiency, and convergence speed—for evaluating and optimizing multi-agent system collaboration performance.

### C. Prompt Engineering Strategy

For the specialized nature of PLC code generation, this research has designed a hierarchical prompt engineering strategy to fully leverage LLMs' code generation capabilities. The foundational layer employs role-based prompting, positioning the model as an experienced PLC programming expert in industrial control, enhancing the model's understanding and application of domain knowledge. The task layer adopts structured task descriptions, decomposing code generation tasks into four stages: requirement understanding, architecture design, code implementation, and testing verification, with each stage accompanied by specific input-output specifications and example code. The context layer dynamically injects PLC programming standards, platform-specific documentation, and historical code snippets relevant to current tasks through Retrieval-Augmented Generation (RAG) technology, providing rich contextual information for the model. The constraint layer defines strict constraints for code generation, including adherence to IEC 61131-3 standards, avoidance of deprecated functions, ensuring type safety, and meeting real-time requirements. The feedback layer designs an iterative optimization mechanism that generates targeted improvement prompts based on compiler feedback, verification tool output, and performance test results, guiding the model to correct errors and optimize code. This multi-level prompting strategy effectively enhances code generation accuracy, reliability, and platform adaptability.

The prompt template design follows the general structure:

$$P = \langle R, T, C, E, F \rangle \tag{2}$$

where $R$ represents role-based prompts, $T$ represents task descriptions, $C$ represents contextual information, $E$ represents example code, and $F$ represents constraint conditions. The complete prompt $P$ forms structured input for the LLM through concatenating these five components.

### D. Code Verification Mechanism

To ensure the reliability and safety of generated PLC code in actual industrial environments, this research establishes a multi-level code verification mechanism. The first layer involves syntax verification, utilizing PLC programming language parsers to check lexical and syntactic correctness, promptly detecting spelling errors, syntax errors, and formatting issues. The second layer involves static analysis, employing data flow and control flow analysis techniques to detect static defects such as uninitialized variables, dead code, infinite loops, and potential null pointer references. The third layer involves type checking, verifying type consistency of variables, function parameters, and return values to prevent runtime errors caused by type mismatches. The fourth layer involves logic verification, adopting model checking techniques to transform PLC code into state transition systems, describing system properties using temporal logic formulas, and verifying whether code satisfies safety and liveness properties through symbolic model checkers (such as NuSMV). The fifth layer involves simulation testing, running generated code in virtual PLC environments, simulating various normal and abnormal operating conditions to verify code functional correctness and robustness. In robotic arm refractory brick handling application scenarios, the verification mechanism particularly focuses on the correctness of critical functions including position accuracy, force control, collision detection, and emergency stop.

The formal definition of code correctness can be expressed as:

$$\text{Correct}(P) = \text{Syntax}(P) \wedge \text{Type}(P) \wedge \text{Logic}(P) \wedge \text{Safety}(P) \tag{3}$$

where $P$ represents the PLC program under verification, $\text{Syntax}(P)$ represents syntax correctness, $\text{Type}(P)$ represents type correctness, $\text{Logic}(P)$ represents logic correctness, and $\text{Safety}(P)$ represents safety properties. Code is considered correct only when all conditions are simultaneously satisfied.

### E. Domain Knowledge Injection Mechanism

Effective injection of domain knowledge is crucial for improving PLC code generation quality. This research adopts a hybrid strategy combining Retrieval-Augmented Generation (RAG) and Parameter-Efficient Fine-Tuning (PEFT). The RAG module constructs a vector database containing PLC programming manuals, standard function libraries, platform documentation, and historical code repositories, employing a hybrid retrieval strategy combining dense vector retrieval and sparse keyword retrieval to improve retrieval accuracy and recall [12]. The retrieval process adopts a two-stage ranking mechanism: first utilizing vector similarity for coarse ranking, then employing a cross-attention-based reranking model for fine ranking, ensuring that the most relevant knowledge is injected into the generation process. The PEFT module employs Low-Rank Adaptation (LoRA) techniques for domain adaptation of LLMs. While maintaining original model parameters unchanged, only a small number of low-rank matrices are trained, substantially reducing fine-tuning computational and storage overhead. The fine-tuning dataset includes manually annotated PLC code generation task pairs covering mainstream PLC platforms such as Siemens, Rockwell, and Schneider, as well as typical application scenarios including robot control, process control, and motion control. Through the synergistic effect of RAG and PEFT [13], the model can both utilize rich information from external knowledge bases and learn domain-specific programming patterns and best practices, significantly

improving code generation quality and cross-platform adaptation capabilities.

The calculation formula for RAG retrieval relevance scores is:

$$\text{Score}(q,d) = \lambda \cdot \text{Sim}_{dense}(q,d) + (1-\lambda) \cdot \text{Sim}_{sparse}(q,d) \quad (4)$$

where $q$ represents the query vector, $d$ represents the document vector, $\text{Sim}_{dense}$ represents cosine similarity of dense vectors, $\text{Sim}_{sparse}$ represents BM25-based sparse retrieval scores, and $\lambda \in [0,1]$ is the balance coefficient. Final retrieval results are sorted in descending order by score, with the top-K documents injected as contextual information into the generation process.

## IV. Experimental Design

### A. Dataset Construction

To comprehensively evaluate the proposed multi-agent PLC code generation system, this research constructed a comprehensive dataset covering multiple industrial application scenarios, named PLC-MultiTask. The dataset includes three main categories: basic control tasks, complex industrial scenarios, and cross-platform adaptation tasks. Basic control tasks cover fundamental programming patterns including sequential control, conditional control, loop control, timer control, and counter control, collecting 150 annotated samples. Complex industrial scenarios include robotic arm control, conveyor belt systems, filling production lines, temperature control systems, and material handling systems, collecting 200 annotated samples, with 50 samples dedicated to robotic arm refractory brick handling scenarios. Cross-platform adaptation tasks target mainstream PLC platforms including Siemens S7-1200/1500, Rockwell ControlLogix, Schneider Modicon M340, and CODESYS, collecting 80 platform-specific code samples for each platform. The dataset construction process includes five stages: requirement description collection, manual code writing, expert review, platform testing verification, and quality annotation, ensuring each sample possesses accurate requirement descriptions, executable reference code, and detailed functional annotations. All code adheres to the IEC 61131-3 standard and has passed functional testing in corresponding PLC simulation environments.

### B. Baseline Methods

To comprehensively evaluate the effectiveness of the proposed method, five baseline methods were selected for comparative experiments. The first is the Pure Large Language Model method (Pure-LLM), directly using GPT-4 and CodeLLaMA-34B models for zero-shot code generation without any domain adaptation or knowledge enhancement. The second is the Prompt-Enhanced method, applying carefully designed prompt templates to Pure-LLM, including role positioning, task descriptions, and example code, but without using RAG or multi-agent mechanisms. The third is the RAG-Only method, employing a single LLM combined with RAG technology, retrieving relevant information from domain knowledge bases to assist code generation but not adopting multi-agent collaboration. Through systematic comparison with these baseline methods, we can fully validate the effectiveness and superiority of the proposed multi-agent collaboration mechanism, prompt engineering strategy, and domain knowledge injection method.

### C. Evaluation Metrics

This research adopts a multi-dimensional evaluation metric system to comprehensively measure PLC code generation system performance. Syntax correctness metrics include Compilation Success Rate (CSR) and Syntax Error Rate (SER) to evaluate basic syntax correctness of generated code. Semantic similarity metrics employ CodeBLEU [14] and BLEU scores to measure similarity between generated code and reference code in syntax structure and semantic expression. Functional correctness metrics include Test Pass Rate (TPR) and Function Completeness (FC), validating the degree of functional implementation of generated code through predefined test case sets. Code quality metrics include Cyclomatic Complexity (CC), Lines of Code (LOC), and Maintainability Index (MI), evaluating structural quality and maintainability of generated code. Efficiency metrics include Generation Time (GT), Iteration Count (IC), and Verification Time (VT), measuring system execution efficiency. Cross-platform adaptability metrics evaluate portability and compatibility of generated code across different PLC platforms, quantified using Platform Adaptation Success Rate (PASR). Notably, although BLEU and CodeBLEU are widely used evaluation metrics in code generation, research has shown these metrics have certain limitations in correlation with human judgment. Therefore, this research simultaneously adopts functional correctness metrics (such as test pass rate) as supplements to more comprehensively evaluate code quality [15]. These metrics constitute a comprehensive evaluation system across four dimensions: correctness, quality, efficiency, and adaptability.

The calculation formula for CodeBLEU is:

$$\text{CodeBLEU} = \alpha \cdot \text{BLEU} + \beta \cdot \text{BLEU}_{weight} + \gamma \cdot \text{Match}_{AST} + \delta \cdot \text{Match}_{DF} \quad (5)$$

where BLEU is the standard BLEU score, $\text{BLEU}_{weight}$ is the weighted BLEU score (assigning higher weights to keywords), $\text{Match}_{AST}$ is the abstract syntax tree matching degree, $\text{Match}_{DF}$ is the data flow matching degree, and $\alpha$, $\beta$, $\gamma$, $\delta$ are weight coefficients satisfying $\alpha + \beta + \gamma + \delta = 1$.

### D. Experimental Results

Table I presents overall performance comparisons of different methods on the PLC-MultiTask dataset. As shown, the proposed multi-agent system achieves optimal performance across all metrics. In compilation success rate, our method reaches 90.3%, improving by 43.3 percentage points compared to the Pure-LLM method. In test pass rate, our method achieves 87.6%, significantly outperforming all baseline methods, demonstrating effective assurance of generated code functional correctness. The CodeBLEU score reaches 75.4,

TABLE I
PERFORMANCE COMPARISON OF DIFFERENT METHODS ON PLC-MULTITASK DATASET

| Method | CSR (%) | TPR (%) | CodeBLEU | BLEU | GT (s) | IC | PASR (%) |
|---|---|---|---|---|---|---|---|
| Pure-LLM (GPT-4) | 47.0 | 38.2 | 54.3 | 48.7 | 12.4 | 1.0 | 52.6 |
| Pure-LLM (CodeLLaMA) | 52.8 | 42.6 | 58.9 | 52.3 | 10.8 | 1.0 | 48.3 |
| Prompt-Enhanced | 74.5 | 68.3 | 66.7 | 61.2 | 15.6 | 2.3 | 68.9 |
| RAG-Only | 81.7 | 76.2 | 72.2 | 67.8 | 19.4 | 2.8 | 74.5 |
| **Ours (Multi-Agent)** | **90.3** | **87.6** | **75.4** | **71.2** | **23.7** | **3.2** | **85.2** |

Note: CSR - Compilation Success Rate, TPR - Test Pass Rate, GT - Generation Time, IC - Iteration Count, PASR - Platform Adaptation Success Rate

TABLE II
PERFORMANCE BREAKDOWN OF DIFFERENT METHODS ON THREE TASK CATEGORIES

| Method | Basic Control CSR/TPR (%) | Complex Industrial CSR/TPR (%) | Cross-Platform PASR (%) | Robotic Arm FC (%) |
|---|---|---|---|---|
| Pure-LLM (GPT-4) | 64.7/58.0 | 38.5/28.0 | 52.6 | 32.0 |
| Prompt-Enhanced | 82.0/76.7 | 61.5/52.5 | 64.8 | 48.0 |
| RAG-Only | 85.3/80.7 | 72.0/65.5 | 70.1 | 68.0 |
| **Ours (Multi-Agent)** | **90.7/86.0** | **82.5/75.3** | **79.2** | **85.0** |

Note: CSR - Compilation Success Rate, TPR - Test Pass Rate, PASR - Platform Adaptation Success Rate, FC - Function Completeness

improving by 3.2 percentage points compared to the RAG-Only method, indicating that the multi-agent collaboration mechanism better captures code syntax and semantic features. Regarding generation time, our method averages 23.7 seconds, slightly higher than the Pure-LLM method, but considering the significantly improved code quality, this time overhead is acceptable. The platform adaptation success rate reaches 85.2%, demonstrating the effectiveness of the cross-platform adaptation mechanism. These results fully validate the superiority of the proposed method, particularly demonstrating clear advantages in complex industrial scenarios and cross-platform adaptation tasks.

Table II presents detailed performance breakdown of different methods across three task categories. We observe that our method achieves 90.7% compilation success rate and 86.0% test pass rate on basic control tasks, approaching perfect performance, indicating the system handles basic programming patterns well. In complex industrial scenario tasks, our method achieves 82.5% compilation success rate and 75.3% test pass rate, still maintaining significant advantages over baseline methods, demonstrating the effectiveness of multi-agent collaboration mechanisms in handling complex control logic. In cross-platform adaptation tasks, our method's platform adaptation success rate reaches 79.2%, notably higher than other methods, indicating the system effectively handles differences between PLC platforms, generating code with good portability. Particularly noteworthy, in the most challenging robotic arm refractory brick handling scenario, our method's function completeness reaches 85.0%, fully validating the system's practical value in real industrial applications. These results demonstrate that the proposed method not only outperforms existing methods in overall performance but also maintains consistent advantages across all subdivided task categories.

## V. CASE STUDY: ROBOTIC ARM REFRACTORY BRICK HANDLING SYSTEM

### A. Application Scenario Description

As critical materials for high-temperature industrial furnaces, refractory bricks face harsh working environments including high temperature, high dust, and high intensity during production and handling processes. Traditional manual handling methods are not only inefficient but also pose serious occupational health risks, with workers experiencing long-term exposure to high-temperature environments easily developing conditions such as heat stroke. To address this issue, a large refractory materials manufacturing enterprise introduced an automated robotic arm handling system requiring the system to accurately grasp refractory bricks weighing 5-50 kilograms in 1200°C high-temperature environments, transferring them from production lines to designated stacking positions while ensuring handling process safety and brick integrity. The system's control logic is extremely complex, requiring coordination of six-degree-of-freedom robotic arm precise motion control, real-time force sensor feedback processing, collision detection and avoidance, temperature monitoring and alarming, and interlock protection with upstream and downstream production equipment subsystems. As the core controller, the PLC must achieve millisecond-level real-time response while meeting strict requirements of industrial safety standards. This application scenario poses extremely high challenges for PLC code generation systems, requiring not only functional correctness but also real-time performance and safety assurance, making it an ideal test case for validating intelligent code generation technology practicality.

### B. System Implementation and Code Generation

For the complex requirements of robotic arm refractory brick handling, the proposed multi-agent system demonstrated excellent code generation capabilities. The Requirement Analysis Agent first conducted deep parsing of user-provided natural language requirements, extracting key control elements including six-axis robotic arm motion trajectory planning, gripper opening/closing control, force sensor threshold settings, collision detection trigger conditions, temperature monitoring alarm logic, and handshake signals with conveyor belt systems. The system identified that this task involves 12 input signals (including start button, emergency stop button, position sensors, force sensors, temperature sensors, etc.) and 18 output signals (including motor control, pneumatic valve control, indicator lights, alarms, etc.), automatically generating detailed functional specifications and state transition diagrams. Based on functional specifications, the Code Generation Agent retrieved relevant motion control templates, force control algorithms, and safety interlock logic from the domain knowledge base, generating a complete PLC program including main program, motion control subroutines, force control subroutines, collision detection subroutines, and exception handling subroutines, totaling approximately 800 lines of Structured Text code. The Verification Agent conducted multi-level verification of generated code, passing syntax checking, type checking, and logic verification, successfully passing 36 test cases in virtual PLC environments including normal handling processes, collision detection, force overload protection, and emergency stop. The Optimization Agent further optimized the code, eliminating three redundant calculations, optimizing motion trajectory smoothness, and performing specific adaptation for the Siemens S7-1500 platform. The finally generated code operates stably on actual PLC controllers, meeting all functional and performance requirements.

### C. Experimental Verification and Performance Analysis

To validate generated code effectiveness in actual industrial environments, month-long testing was conducted at the enterprise production site. During testing, the robotic arm system operated cumulatively for 720 hours, completing over 28,000 refractory brick handling tasks with a 99.2% success rate. Only 22 failures due to brick damage-caused grasping failures and 1 safety shutdown triggered by communication delays occurred, far below the enterprise-required 5% failure rate threshold. The system's average handling cycle was 7.8 seconds, improving 5.8 times compared to the manual handling average of 45 seconds, substantially increasing production efficiency. Regarding safety, the system's collision detection response time was 8 milliseconds, and emergency stop response time was 15 milliseconds, both meeting industrial safety standard requirements. The temperature monitoring system successfully intercepted three high-temperature anomaly events, preventing equipment damage and safety accidents. Regarding code quality, the generated PLC program received an average rating of 8.5 out of 10 from professional engineer reviews, with engineers considering the code well-structured, logically rigorous, and thoroughly commented, meeting industrial application quality standards. Table III presents detailed comparisons of code generated by our method with manually written code and other automatic generation methods in this case. As shown, our method achieves ideal balance in functional implementation, code quality, and development efficiency, validating the feasibility and practical value of the multi-agent PLC code generation system in real industrial applications.

TABLE III
CODE QUALITY COMPARISON IN ROBOTIC ARM REFRACTORY BRICK HANDLING CASE

| Dimension | Manual | Ours | Pure-LLM |
|---|---|---|---|
| Development Time (hrs) | 120 | 32 | 8 |
| Lines of Code (LOC) | 1050 | 820 | 650 |
| Compilation Success | 100% | 100% | 45% |
| Test Pass Rate | 100% | 97.2% | 38.9% |
| Code Quality Score | 9.5/10 | 8.5/10 | 4.2/10 |
| Maintainability Index | 88 | 82 | 52 |
| Success Rate (Field) | 99.8% | 99.2% | N/A |

### D. Limitation Analysis

Although the proposed multi-agent PLC code generation system has achieved positive results in industrial applications, several limitations require further improvement. In complex motion trajectory optimization, generated code efficiency has not reached manual optimization levels when handling multi-constraint planning problems, requiring integration of professional motion planning algorithms, while the system's conservative exception handling strategy tends to trigger safety shutdown mechanisms rather than implementing intelligent fault tolerance, affecting system availability and necessitating enhanced fault diagnosis and recovery capabilities. From language support perspective, the system has primarily optimized Structured Text with insufficient support for other IEC 61131-3 programming languages such as Ladder Diagram and Function Block Diagram, limiting applicability across different scenarios, while domain knowledge base maintenance faces challenges in covering new PLC platforms and emerging applications, directly impacting code generation quality and requiring more efficient knowledge update mechanisms. Multi-agent collaboration mechanisms, while improving code quality, correspondingly increase computational overhead and system complexity, potentially creating deployment difficulties in resource-constrained edge computing environments, and the system's limited interpretability struggles to provide clear causal analysis and specific improvement paths when code generation fails or quality issues arise, posing challenges for enhancing engineer user experience.

## VI. CONCLUSION

This research addresses the complexity of PLC programming in industrial automation and the high costs of manual programming by proposing a multi-agent-based cross-platform intelligent PLC code generation method. The method constructs a collaborative system comprising requirement analysis agents, code generation agents, verification and testing agents,

and optimization agents. Combined with Retrieval-Augmented Generation technology and Parameter-Efficient Fine-Tuning strategies, it achieves end-to-end automated generation from natural language requirements to executable PLC code. The system designs hierarchical prompt engineering strategies, establishes multi-level code verification mechanisms, and achieves unified support for mainstream PLC platforms including Siemens, Rockwell, and Schneider. Experiments on the PLC-MultiTask comprehensive dataset demonstrate that our method significantly outperforms existing methods across key metrics including compilation success rate, test pass rate, CodeBLEU score, and platform adaptation success rate. In a real industrial case of robotic arm refractory brick handling, system-generated code underwent one month of field testing validation, achieving 99.2% success rate and reducing development time by 73.3% compared to manual programming, fully demonstrating the method's practicality and reliability.

This research constructs a multi-agent collaborative architecture for PLC code generation, achieving inter-agent information exchange and task coordination, effectively improving code generation accuracy and reliability, providing a novel technical paradigm for intelligent code generation of domain-specific languages. The designed domain knowledge injection mechanism integrating RAG and PEFT significantly enhances LLMs' understanding and application capabilities for PLC programming knowledge, providing valuable reference for code generation research in low-resource programming languages. The established comprehensive code verification system conducts multi-level verification from syntax, type, logic to safety, ensuring generated code reliability and safety in industrial environments. Implementation of the cross-platform adaptation mechanism supports unified code generation for multiple mainstream PLC platforms, reducing equipment migration costs and technical barriers for industrial enterprises. Through real industrial case validation, this method demonstrates excellent engineering practical value, accumulating valuable experience for intelligent code generation technology implementation in industrial domains. Future research will further enhance system multi-modal understanding capabilities, reinforcement learning mechanisms, and lightweight deployment capabilities, promoting PLC programming toward higher levels of intelligence and automation, providing stronger technical support for Industry 4.0 and smart manufacturing.

Future work will advance the proposed system through several interconnected directions. The research will enhance multi-modal understanding capabilities to enable direct PLC code generation from graphical representations such as P&IDs, timing diagrams, and state machine diagrams, while incorporating reinforcement learning mechanisms for autonomous optimization. System capabilities will expand to support all five IEC 61131-3 programming languages with automatic conversion functionality, explore mixed programming modes integrating high-level languages, and develop intelligent code review modules for automatic detection of performance bottlenecks and safety hazards. An open community and knowledge sharing platform will aggregate industrial cases and best practices, while lightweight multi-agent collaboration mechanisms will enable edge-side intelligent code generation. Finally, strengthening human-machine collaboration through intuitive interaction interfaces will support engineers in efficiently reviewing and tuning generated code, achieving deep integration of human expertise with artificial intelligence capabilities.

## REFERENCES


[1] M. Seitz, F. Gehlhoff, L. A. Cruz Salazar, A. Fay, and B. Vogel-Heuser, "Automation platform independent multi-agent system for robust networks of production resources in industry 4.0," *Journal of Intelligent Manufacturing*, vol. 32, no. 7, pp. 2023–2041, Jul. 2021. DOI: 10.1007/s10845-021-01759-2

[2] O. Rashad, O. Attallah, and I. Morsi, "A smart PLC-SCADA framework for monitoring petroleum products terminals in industry 4.0 via machine learning," *Measurement and Control*, vol. 55, no. 7-8, pp. 830–848, Aug. 2022. DOI: 10.1177/00202940221103305

[3] H. Koziolek, S. Gruener, and V. Ashiwal, "ChatGPT for PLC/DCS control logic generation," in *Proc. IEEE 28th Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sinaia, Romania, Sep. 2023, pp. 1–8. DOI: 10.1109/ETFA54631.2023.10275411

[4] 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 industrial control systems," in *Proc. 46th Int. Conf. Softw. Eng.: Softw. Eng. Pract. (ICSE-SEIP)*, Lisbon, Portugal, Apr. 2024, pp. 192–203. DOI: 10.1145/3639477.3639743

[5] Z. Liu, R. Zeng, D. Wang, G. Peng, J. Wang, Q. Liu, P. Liu, and W. Wang, "Agents4PLC: Automating closed-loop PLC code generation and verification in industrial control systems using LLM-based agents," arXiv preprint arXiv:2410.14209, Oct. 2024. DOI: 10.48550/arXiv.2410.14209

[6] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 8, pp. 1–79, Nov. 2024. DOI: 10.1145/3695988

[7] S. Joel, J. Wu, and F. Fard, "A survey on LLM-based code generation for low-resource and domain-specific programming languages," *ACM Trans. Softw. Eng. Methodol.*, 2024. DOI: 10.1145/3770084

[8] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, and J.-R. Wen, "A survey on large language model based autonomous agents," *Front. Comput. Sci.*, vol. 18, no. 6, Article 186345, Mar. 2024. DOI: 10.1007/s11704-024-40231-1

[9] J. Lim, B. Vogel-Heuser, and I. Kovalenko, "Large language model-enabled multi-agent manufacturing systems," in *Proc. IEEE 20th Int. Conf. Autom. Sci. Eng. (CASE)*, Bari, Italy, Aug. 2024, pp. 3940–3946. DOI: 10.1109/CASE59546.2024.10711432

[10] A. Sidorenko, W. Motsch, M. van Bekkum, N. Nikolakis, K. Alexopoulos, and A. Wagner, "The MAS4AI framework for human-centered agile and smart manufacturing," *Frontiers in Artificial Intelligence*, vol. 6, Article 1241522, Sep. 2023. DOI: 10.3389/frai.2023.1241522

[11] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber, "MetaGPT: Meta programming for multi-agent collaborative framework," arXiv preprint arXiv:2308.00352, Aug. 2023. DOI: 10.48550/arXiv.2308.00352

[12] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," arXiv preprint arXiv:2312.10997, Mar. 2024. DOI: 10.48550/arXiv.2312.10997

[13] Y. Wu, Y. Zhao, B. Hu, P. Minervini, P. Stenetorp, and S. Riedel, "An efficient memory-augmented transformer for knowledge-intensive NLP tasks," arXiv preprint arXiv:2210.16773, Oct. 2022. DOI: 10.48550/arXiv.2210.16773

[14] S. Ren, D. Guo, S. Lu, L. Zhou, S. Liu, D. Tang, N. Sundaresan, M. Zhou, A. Blanco, and S. Ma, "CodeBLEU: A method for automatic evaluation of code synthesis," arXiv preprint arXiv:2009.10297, Sep. 2020. DOI: 10.48550/arXiv.2009.10297

[15] M. Evtikhiev, E. Bogomolov, Y. Sokolov, and T. Bryksin, "Out of the BLEU: How should we assess quality of the code generation models?" *J. Syst. Softw.*, vol. 203, Article 111741, Sep. 2023. DOI: 10.1016/j.jss.2023.111741