HUNGARIAN JOURNAL OF
INDUSTRY AND CHEMISTRY
Vol. 54(SI) pp. 35–46 (2026)
hjic.mk.uni-pannon.hu
DOI: 10.33927/hjic-2026-15

HUNGARIAN JOURNAL OF INDUSTRY AND CHEMISTRY
HJIC

RECENT ADVANCES IN PLC PROGRAMMING USING ARTIFICIAL INTELLIGENCE AND LARGE LANGUAGE MODELS

PÉTER BÁLINT MEZŐ[1]* AND ÁDÁM JACSÓ[1]

1 Department of Manufacturing Science and Engineering, Faculty of Mechanical Engineering, Budapest University of Technology and Economics, Műegyetem rkp. 3., 1111 Budapest, HUNGARY

Integrating Artificial Intelligence (AI) into process control is one of the most significant technological trends in industrial automation today. The generative programming of Programmable Logic Controllers (PLCs) is a prominent example of this development. While Artificial Neural Networks (ANNs) have previously been applied for tasks such as natural language processing and fault prediction in PLC hardware, recent advancements in Large Language Models (LLMs) have further expanded AI capabilities, enabling the interpretation of complex prompts and assisting with control code generation. Development tools and industrial copilots powered by generative AI are increasingly being proposed to support engineers in managing control systems, with the potential to simplify and accelerate control software development. In contrast to these promises, using generative models in PLC programming is still in its early stages, characterized by exploratory research and cautious implementation. This review provides a systematic overview of recent developments in AI-assisted PLC programming, focusing on generative approaches. It synthesizes emerging methodologies, tools, and applications while critically examining current limitations and outlining potential research directions in industrial control systems.

Keywords: Automatic PLC programming, Generative AI, LLM, PLC, ANN

1. Introduction

Automation has become essential for maintaining competitiveness in today's rapidly evolving industry. As industries encounter ever-increasing challenges, the need for modular, interchangeable systems and enhanced physical processes has become crucial. One of the major focal points lies in the automatic programming of Programmable Logic Controllers (PLCs). By reducing the reliance on manual coding, this approach accelerates system deployment, enhances flexibility, and lowers the barriers to implementation, making automation more accessible and efficient.

Since the advent of LLMs, the use of Generative AI has been increasing, and AI-based tools have gained widespread attention and demonstrated significant potential across various domains. Tasks have become possible in a short time, from explaining documents to extracting bullet points from lengthy texts. With this, their capability to understand programming languages or follow instructions has been attracting attention. At the Hannover Fair 2023, Siemens and Microsoft introduced conceptual frameworks and ideas for a generative AI-powered assistant [1]. This collaboration led to the public launch of Siemens Industrial Copilot at the 2024 Hannover Fair. It is an AI-based support tool aimed at assisting professionals across many areas of the industrial value chain, from designing and planning to operations and service. The system combines Siemens's industrial "know-how" and expertise with Microsoft's Azure OpenAI Service. Other major industrial players began introducing similar solutions following this announcement. Building on their collaboration in 2020, Rockwell Automation and Microsoft introduced their own Industrial Copilot in 2024 [2],[3]. Similarly, in partnership with Microsoft, Schneider Electric launched its version of an AI Copilot in May 2025 [4],[5]. These copilots are powered by LLMs, which are increasingly capable of handling a broad set of auxiliary tasks, such as document summarization, and even analyzing or suggesting modifications to control logic. While these solutions serve as supportive tools and downloadable plug-ins for applications rather than integrated autonomous control agents, they signal a shift in industrial settings toward a human-AI collaboration.

This review aims to explore the recent developments in the application of Artificial Intelligence, especially Generative AI and LLMs, for PLC programming. Section 2 provides a brief introduction to programmable logic controllers and artificial intelligence. Section 3 reviews state-of-the-art research efforts on AI-assisted PLC programming applications

Received: 12 Nov 2025; Revised: 19 Dec 2025;
Accepted: 20 Dec 2025
*Correspondence: mezopeterbalint@edu.bme.hu

and theories. Section 4 categorizes articles by application, and Section 5 identifies limitations, summarizes findings, and suggests future directions. As the paper covers a broad range of subjects, *Table 1* provides an overview of the addressed topics and articles.

2. Background and needs in PLC programming

Programmable Logic Controllers (PLCs) are the backbone of industrial automation. They utilize various programming languages based on textual and/or graphical methods for their function. All these programming languages adhere to the IEC 61131-3 standard. This widely adopted standard enhances consistency, however, it is not legally binding and may not guarantee complete portability due to the manufacturer's proprietary hardware and software setups. Textual languages are commonly used and include Structured Text (ST) and Instruction List (IL). ST is akin to high-level programming languages, as it supports the use of loops and conditional constructs for complex programming strategies. In contrast, unless modified by brackets, IL language uses direct derivation from Boolean equations through mnemonic instructions following the "AND before OR" rule. Graphical methods are used to make a complex control logic easier to understand and modify, such as the Sequential Function Chart (SFC), Ladder Diagram (LD), and Function Block Diagram (FBD). SFC visualizes control steps and transitions, while LD mimics electrical relay schematics, making it intuitive for maintenance personnel. FBD is used for efficient task control and monitoring, with its construct being based on connecting predefined function blocks [30],**Hiba! A hivatkozási forrás nem található.**.

A critical phase in PLC software development is accurately defining system requirements, since many errors arise from inadequate specifications. Developers must consider operational behavior, human-machine interfaces, and the integration of existing modules. Testing methods ensure the validation of control logic and early fault detection. These tests can be hardware-in-the-loop (HiL) or software-in-the-loop (SiL) [30].

Writing a PLC program can rely heavily on specialized domain knowledge, be labor-intensive, and prone to errors. The development must adhere to several constraints, such as the lack of universal PLC programming software across PLCs. Manual coding and testing are becoming increasingly challenging due to the growing complexity of systems and customized setups, which creates an underlying need for automation in PLC programming to improve code quality, maintainability, and development efficiency. Recent solutions promise to address this issue by generating programs, supporting test case generation, or fulfilling specific domain requirements with AI.

However, challenges persist for such solutions. One limitation is the lack of sufficient training data, as correct, working code is mostly undisclosed. The challenge is compounded by variations in vendor implementations, as programs cannot be exchanged across PLCs from different vendors. Another difficulty is the absence of standardized evaluation metrics. In the case of rigorous security specifications, some generated programs could contain flaws that can become hidden traps (e.g. instantaneous shutdown may not work in a fail-safe environment or fail-operational mode may not work). To address these issues, this paper examines the possibilities of fitting Generative AI with domain-aligned prompt engineering, retrieval-augmented generation, and fine-tuning to enable scalable, AI-assisted software engineering for PLCs.

3. Methods and algorithms to automate PLC programming

This section will discuss the research activities and solutions for AI-aided PLC programming. In the first subsection, control logic programming is discussed, using traditional algorithms to overcome challenges. The second subsection examines the usage of Artificial Neural Networks (ANNs), and the third will discuss the solutions derived from general-purpose and domain-specific LLMs.

3.1. Traditional algorithms

The application of general models is a pivotal area of research, offering advances in operational efficiency and adaptability in industrial automation. This section explores various innovative approaches and examines a range of creative methodologies that employ traditional models to address intricate challenges inherent in automation processes. For example, control systems integrating computer vision and evolutionary algorithms optimize process control underscore the transformative potential of general models in enhancing industrial operations.

Hu et al. addressed the lack of formal semantics and vendor-specific inconsistencies in ST programming by developing ST-Petri. Based on Petri nets, this visual executable semantic model formalizes ST constructs such as data types, control structures, and function calls while visualizing process control through graphs. They validated ST-Petri by executing a test set of real and mutated ST programs and comparing the results to those from OpenPLC. The experiments showed improved compilation pass rates and made error reporting more transparent than OpenPLC's. Although the model is not complete, it shows promise for handling larger programs and enabling a more intuitive understanding of control logic. The authors noted support for timing aspects and PLC-specific environments to facilitate parallel compilation as future development directions [6].

Cai et al. addressed the time-consuming, error-prone nature of manual programming of mode-based control algorithms in building automation by developing a software-assisted framework for automatic PLC code generation. The solution is based on their earlier MODI

Table 1: Summary of the reviewed articles

Topic	Theme	Authors	Methodology and Results
ST-Code Generation	Formal semantic modelling of PLC ST using Petri nets	Hu et al. [6]	• This project improves PLC software reliability by addressing the lack of formal semantics and vendor inconsistencies in ST programming. • Developed ST-Petri, a Petri net-based semantic model with visual control flow, validated through execution tests on GitHub code. It outperformed OpenPLC with higher compilation pass rates and clearer error reporting, offering explicit semantics and scalability for large programs.
	Automatic PLC code generation	Cai et al. [7]	• Reducing manual programming effort and errors in building automation. Developed framework based on the MODI method, translated algorithms into ST code, and tested via simulation. • Successfully demonstrated intended control behaviour in a case study, reduced implementation effort and errors, with plans to integrate digital data sources for broader deployment.
	Enhancing productivity and efficiency in control logic development	Koziolek et al. [8]	• Created 100 prompts across ten task categories, introduced a reusable prompt library. Results show GPT-4 can generate syntactically correct ST code, propose hypotheses on productivity gains, and prompt engineering efficiency.
	Automating PLC code generation using LLMs and retrieval-based augmentation	Koziolek et al. [9]	• Automating PLC code generation using LLMs and retrieval-based augmentation. • Used GPT-4, LangChain, OpenPLC, OSCAT library; retrieval-augmented prompt formulation via similarity search. • Successfully generated and simulated valid ST code; reduced programming time; identified future improvements.
	Addressing the lack of standardisation and customisation in ST programming	Yang et al. [10]	• Developed libraries for mapping requirements to code, a retrieval module, and an extensible code checker • Outperformed state-of-the-art baselines in compilation success rates and fewer errors.
	Improving automation capabilities in modern production systems	Xia et al. [11]	• Proposed a multi-dimensional classification framework, multimodal-driven generation models. • Improved accuracy, adaptability, and code generation efficiency.
	Evaluating advanced LLMs for industrial control logic	Tran et al. [12]	• Systematically benchmarked five LLMs, including ChatGPT-4, using a multi-step prompt engineering strategy. • ChatGPT-4 outperformed other models in syntactic and functional accuracy.
	Addressing limitations of conventional supervisory control methods	Boudribila et al. [13]	• Utilised Reinforcement Learning and NLP techniques, including LLMs and NER. • Demonstrated potential of generative models in interpreting requirements and generating control code, with future research needed on code validation and real-world deployment.
	Automating the translation of P&IDs into control code	Koziolek et al. [14]	• Proposed LLM-based workflow evaluated using GPT-4V across industrial case studies. • Demonstrated feasibility of generating control logic from P&IDs, with future enhancements in prompt refinement and integration of vendor-specific libraries.
	Demonstrates how LLMs can support engineers across the entire PLC software lifecycle	Reinhardt et al. [15]	• Implementation of a local language model for ST code generation; integration of a RAG system • Showed potential in automating specification writing, HMI layout design, ST code generation, testing, and documentation.
	Automated generation and verification of PLC code	Liu et al. [16]	• Bridging the gap between natural language specifications and verifiable control logic Retrieval-Augmented Generation, chain-of-thought prompting, specialised prompt engineering. • Outperformed existing approaches in automation fidelity and verification robustness.
	MetaIndux-PLC for LLM-based automation in PLC code generation	Ren et al. [17]	• Built ST4Indux dataset, applied the CLIFT method, trained specialised LLM, and introduced multi-dimensional evaluation. • Achieved more accurate and reliable PLC code generation, reduced costs, improved flexibility limited by dataset scale and industrial complexity.
	Applying LLMs to PLC programming using a custom-created prompt dataset.	Zheng et al. [18]	• Developed the PLC-100 dataset, proposed a GPT-4-based generation approach with adaptive evaluation • Demonstrated improved reliability and operability of PLC code; identified future refinement needs

Table 1: Summary of the reviewed articles (continued)

Topic	Theme	Authors	Methodology and Results
ST-Code Generation and Testing	Enhancing LLMs for industrial automation	Haag et al. [19]	• Addressing limited training data and syntactic complexity in ST programming. Iterative, preference-based training strategy with DPO, compiler-based validation. • Improved compilation success rates and semantic precision, demonstrating potential for developing specialised industrial copilots for PLC programming.
	Enhancing the reliability and verifiability of LLM-generated code	Fakih et al. [20]	• Bridging the gap between general-purpose LLMs and ICS programming. • User feedback, external validation mechanisms, Prompt Engineering, LoRA fine-tuning. • Improved code generation success rates and expert-rated code quality scores, highlighting potential for verifiably correct automation software.
FBD Code Generation	Streamlining PLC software engineering	Ogundare et al. [21]	• Leveraging no-code/low-code paradigms and machine learning for code synthesis. • Heuristic-driven selection mechanism, context-aware RNN, recommender system. • Confirmed feasibility in industrial process design, illustrating potential for reducing programming overhead and accelerating automation deployment.
FBD Code Tester	Test generation for FBD programs	Liu et al. [22]	• Improving fault detection and reducing computational effort. • Proposed method using MuFBDTester tool and Yices SMT solver, automatic identification of equivalent mutants. • Achieved nearly 100% mutation scores, outperformed structural coverage techniques, proved faster and scalable for larger programs.
	Automating test case generation for control logic	Koziolek et al. [23]	• Addressing limitations of traditional testing methodologies. • LLM to synthesise test inputs and outputs, enhanced prompting strategy. • Achieved high statement coverage for low- to medium-complexity programs.
SFC and LD-Code Generation	Supporting the generation of graphical languages	Zhang et al. [24]	• Addressing the underexplored area of graphical languages in AI-assisted PLC programming. • Prompt engineering, example-driven inputs. • Successfully generated simple SFCs but struggled with LDs.
Structured Graph Generation	Autonomous code generation in process control	Löppenberg et al. [25]	• Addressing the complexity of coupled system dynamics. • Graph-based evolutionary algorithm for inductive link prediction. • Validated on PLC-controlled manufacturing process, showing performance comparable to manually engineered solutions.
SCL or IL Code Generation	Automating industrial process control	Löppenberg et al. [26]	• Adaptability and automation in process control. • Developed a framework using evolutionary algorithms, digital twin, and evaluation function. • Validated on a liquid station setup, demonstrated adaptability and automation.
SCL or XML Code Generation	Reducing reprogramming effort in adaptive manufacturing	Fan Mo et al. [27]	• Integrating ontological modelling, rule-based reasoning, and Graph Neural Networks. • Automated creation, reconfiguration, and validation of PLC programs, reducing human effort and commissioning time. • The model successfully integrates with the PLCOpen XML, demonstrating scalability and compatibility.
BT Introduction to PLC programming	Improving modularity and adaptability in automation	Sidorenko et al. [28]	• Merging low-level device control with high-level task coordination. • Three strategies for integrating BTs into PLC architectures. • Demonstrated feasibility of BT-enhanced PLC programming, offering better abstraction and modularity.
ST and FBD Code to Python Translator	Automated testing for PLCs via code translation	Salari et al. [29]	• Bridges PLC environments with Python-based test generation for safety-critical systems. • Translation of ST/FBD PLC code to Python; validation using behaviour-, rule-, and coverage-based tests. • Demonstrated that PyLC can support unit testing and test generation; future improvements target automation and formal verification.

method, which formalized control logic descriptions using signal-interpreted Petri nets to reduce ambiguity. This new framework automatically translated the formalized algorithms into ST code for PLCs. The generated code was then tested through simulation, and the intended control behavior was successfully demonstrated in a case study involving an air handling unit. While the approach reduces implementation effort and potential errors, current limitations include reliance on manual data input. The authors aim to integrate digital data sources to streamline the process further and enable broader deployment in building automation systems in the future [7].

Löppenberg et al. addressed the challenge of automating PLC-based industrial process control by developing a self-optimizing framework using Evolutionary Algorithms (EA). Motivated to interpret logical relationships in cyclic process data accurately, they proposed a system capable of automatic code generation from optimized solutions. The architecture consisted of three central units (upstream, intermediate, downstream), where a digital twin and evaluation function guided the learning process in a closed loop. The system optimized control logic through redundancy, priority, and sequencing, and was validated on a liquid station setup under multi-objective optimization conditions. The main advantage of the approach was its adaptability and automation, though complexity and the need for domain-specific tuning remained limitations. The authors aim to enhance the method with neural networks and multi-agent systems for broader applicability and improved traceability through a bijective mapping of evolutionary steps [26].

Liu et al. addressed the limitations of structural coverage-based test generation for FBD programs by proposing an automated method to generate mutation-adequate test suites. The solution implemented was a Yices SMT solver within the MuFBDTester tool. After that, they created test data to evaluate whether the solution could detect artificial and real faults by differentiating between original and mutated program outputs. A key benefit of the approach was the automatic identification of equivalent mutants, significantly reducing unnecessary computational effort. This method achieved nearly perfect mutation scores in experiments, outperformed structural coverage techniques in fault detection, and proved over 120 times faster in certain complex cases. The approach also demonstrated strong scalability for larger programs and confirmed the coupling effect hypothesis in FBD testing. The authors proposed that future research include mutation operators, support for multiple SMT solvers, and extend the method's applications to industrial systems [22].

Löppenberg et al. also investigated autonomous program code generation to address the difficulty of modelling and controlling complex coupled systems. They developed a structured graph generation approach using Graph-Based Evolutionary Algorithms (GBEA). In their approach, the task was formulated as an inductive link prediction problem, where system logic was iteratively refined in a closed loop with expert knowledge and constraints. The method produced interpretable, customized PLC control logic for an industrial case study, reaching a quality comparable to professional engineering solutions while improving adaptability and process efficiency. The authors highlighted its transferability to other automation tasks but also noted challenges in system stability and scalability, which define the main directions for further research [25].

Sidorenko et al. addressed the limited flexibility of PLC software in environments where modular and rapidly reconfigured control programs are required. They proposed integrating the Behavior Tree (BT) frameworks into PLCs to separate hardware-specific functions from higher-level coordination logic and to improve modularity. Three integration strategies were developed: one linked PLCopen function blocks with external BT libraries, while the others created PLC-based BT implementations for IEC 61131 and IEC 61499 standards. Their test application showed that BTs enabled more explicit task structuring, faster reconfiguration, and better alignment with supervisory control needs. However, challenges regarding legacy device integration still remained. Furthermore, BTs offered greater modularity, abstraction, and extensibility than the SFC programming language, while complementing state machines at lower levels. The authors emphasized future work on extending skill-based PLC programming and integrating planning methods with BT-driven execution [28].

3.2. Artificial neural networks

Solutions with ANNs are also explored for PLC programming. ANNs can help generate a solution for flawed PLC code by learning patterns from previously accepted or rejected codes, which can help with code completion and error correction.

Ogundare et al. introduced a no-code framework to streamline the PLC software engineering process in industrial automation by generating PLC programs in FBD based on requirement documents. Their method leveraged the interest in no-code/low-code paradigms and machine learning-based code synthesis. They proposed an architecture that translates structured documentation through a heuristic-driven selection mechanism into an FBD program. This mechanism utilizes the output of a recommender system to guide the construction of valid and semantically consistent FBD representations based on context relevance. The architecture is enhanced by a Context-Aware Recurrent Neural Network (CA-RNN). This network is used recursively as a generative model and is trained on customer-specific data. This configuration enables the framework to predict the most likely function block based on the program's current state and optimize the function block selection sequence by utilizing learned probabilistic dependencies. The results of experiments confirm the feasibility of this approach in industrial process design, highlighting improvements in training convergence when domain-specific heuristics are incorporated. The authors note that future directions

include integrating documentation-based code generation with machine learning to minimize programming overhead and enhance automation deployment in PLC-based environments [21].

Fan Mo et al. developed an automated, vendor-agnostic methodology for PLC code generation and testing to address the challenge of frequent and error-prone reprogramming in adaptive manufacturing systems. Their solution integrated ontological modelling with rule-based reasoning and Graph Neural Networks (GraphSAGE) to infer process logic from product requirements. This framework enabled automatic creation and reconfiguration with validation of PLC programs. The system reduced human effort and commissioning time with supported modular code reuse, semantic process mapping, and test generation, both structural and mutation-based. The authors aim to extend the methodology to physical hardware and broaden the compatibility across PLC vendors. However, they note the challenges remaining with real-world integration and safety assurance. The approach demonstrated scalability and compatibility with PLCopen XML but relied on advanced tooling and ontology design. This may limit industrial adoption if standardization does not improve [27].

3.3. Large language models

The advent of pretrained large language models has marked a shift in technology. LLMs are already used in industrial automation, providing support for sophisticated code generation and system integration solutions. This section reviews the applications of LLMs, supporting programming tasks, understanding problems and testing programs. Through rigorous research and empirical implementations, these studies illuminate the transformative capabilities of language models in contemporary automation.

3.3.1. General-purpose language models

LLMs have already shown their potential to understand programming languages and generate code and sequences, prompting further investigation into their capability to understand and produce code for PLCs and DCSs (Distributed Control Systems). If the effort is successful, it could open new opportunities to integrate LLMs into existing industrial applications. This subsection discusses various methodologies that harness these models to produce adaptable and efficient code.

Koziolek et al. conducted an exploratory study to assess the potential of GPT-4-based ChatGPT in supporting control engineers with PLC and DCS programming tasks given the limited exploration of LLMs in control logic development. One hundred prompts in ten categories have been designed and tested to generate ST codes. The results show a strong syntactic accuracy and reasoning ability in ChatGPT, which could lead to potential productivity gains. They proposed the prompt set as a foundation for benchmarking LLMs in automation tasks. Compared with other LLM-based tools like GitHub Copilot and AlphaCode, tCPT-4 was hypothesized to improve domain-specific coding efficiency and outperform traditional information retrieval methods. The study also highlights the importance of training engineers in prompt engineering. The authors noted further possibilities for working on prompt refinement, empirical evaluation of productivity gains, and developing a formal quality scoring system for LLM-generated code [8].

Also, Koziolek et al. addressed the challenge of integrating proprietary function block libraries into automatically generated ST code for industrial control systems. A retrieval-augmented code generation method has been developed using text embeddings and similarity search to enrich GPT-4 prompts with relevant function block information. The approach allowed the reuse of vendor-specific, pre-tested blocks while retaining benefits from the speed and adaptability of LLMs. A prototype made from LangChain, OpenPLC, and the OSCAT libraries showed that a valid PLC program could be generated. This prototype could support batch execution, resulting in significantly reduced programming time, but was limited to only specific formats and libraries. The authors aim to expand its applicability to commercial environments, enhance prompt robustness, and integrate automated test generation for quality assurance in the future [9].

Yang et al. addressed the challenge of generating vendor-specific ST code for PLCs by introducing AutoPLC, a multi-agent framework that automates code generation using an LLM. The problem arises from the lack of standardized documentation and the vendor-specific customization of ST, which complicates manual programming and hinders automation. AutoPLC addresses this issue by developing two key libraries: one that maps requirements to code and another containing public instruction sets. It also includes a retrieval module that helps the LLM generate relevant code snippets. An extensible code checker identifies syntax and semantic errors, enabling iterative self-improvement. In evaluations against three benchmarks, CODESYS ST and two variants of Siemens SCL, AutoPLC outperformed seven state-of-the-art baselines, demonstrating higher compilation success rates and fewer errors. An expert reviewed the generated code and confirmed its practical usefulness. The authors intend to expand the framework's coverage to include more complex logic and hardware integration [10].

Xia et al. examined the integration of intelligent manufacturing with PLCs. They noted the growing need for flexibility and real-time responsiveness in contemporary production systems. They found traditional manual programming to be insufficient, leading to their proposal of a multi-dimensional classification framework for automatically generating ST code. This framework is based on various factors – for example, methodology, input types, and application areas. The authors approached the problem with a deeper integration of domain knowledge and intelligent technologies, which led to enhanced automation capabilities. Key developments in their research include multimodal-driven generation models, combination of

diverse data sources, real-time optimization, and a collaborative architecture. The framework could enhance the generation accuracy, adaptability, and efficiency of PLC code based on it, which relies on expert input and possible reliability issues. The authors note that the future direction of intelligent PLC programming can be advanced by leveraging large language models, knowledge graphs, and retrieval-augmented generation in the future [11].

Tran et al. conducted a comparative study to evaluate the capabilities of universal LLMs in ST code generation. Five state-of-the-art models were systematically benchmarked according to the complexity of developing control software and the availability of advanced LLMs, including ChatGPT-4, across 21 representative PLC programming tasks. To evaluate findings, a framework using a multi-step prompt engineering strategy for industrial control logic was set up, with the evaluation framework including metrics such as CodeBERTScore, pass@k, F3-score, and generation time. The results showed that ChatGPT-4 excelled compared to other models in generating correct and functional PLC programs. However, all LLMs were prone to syntactic inaccuracies and semantic inconsistencies. The authors emphasized future work on expanding the set of prompts, incorporating new versions of LLMs and fine-tuning LLMs with domain-specific PLC code [12].

Boudribila et al. addressed the limitations of conventional supervisory control methods in complex and probabilistic systems, introducing an AI-based framework to automate PLC programming. As traditional Supervisory Control Theory (SCT) approaches are not scalable in complex industrial environments, the authors developed a preliminary pipeline that utilizes Reinforcement Learning (RL) and advanced Natural Language Processing (NLP) techniques, including Named Entity Recognition (NER) and LLMs, to automate PLC programming. ChatGPT was used to translate the specifications into ST code. The authors demonstrated the potential of generative models to interpret the requirements and generate executable PLC code by creating a sample dataset and conducting initial experiments. The system has the potential to assist in coding by suggesting code changes, improving controller supervision, and reducing the required expertise and effort. This study showed the transformative potential of LLMs in programming efficiency, system design, and safety in industrial automation. The authors emphasized the need for further research on code validation, formal verification, and real-world deployment. Future studies are underway to address robustness and generalization across applications [13].

Koziolek et al. introduced a novel method for control logic code generation that leverages the multimodal capabilities of LLMs, enhanced with image recognition to synthesize ST directly from Piping and Instrumentation Diagrams (P&IDs). Addressing a long-standing challenge in industrial automation, the manual translation of schematic process documentation into executable control code, the authors propose an LLM-based workflow that can interpret the topological and semantic content of P&IDs and generate functional code with minimal human intervention. The proposed method was evaluated using GPT-4V across three industrial case studies to demonstrate its fundamental feasibility, despite existing limitations in image recognition accuracy. The system could produce valid control logic code within seconds, thereby underscoring its potential for high-efficiency automation engineering. The authors contend that their approach can significantly surpass traditional co-pilot-style code assistants due to the structured nature of automation requirements, which facilitates batch-mode, prompt-driven code generation. Potential future enhancements include the refinement of prompts, support for additional engineering artefacts such as input/output lists and control narratives, and integration of vendor-specific libraries and control function blocks. Moreover, by applying retrieval-augmented generation and introducing automated plausibility checking, the method can be scaled to large industrial projects, generating control logic, simulation scripts, and human-machine interface code. The study provides a foundational framework for further interdisciplinary research, combining deep learning-based diagram interpretation with domain-specific code synthesis in industrial automation [14].

Reinhardt et al. investigated how generative AI can support PLC developers throughout the entire software lifecycle. Their study focused on implementing a local language model for generating ST code and integrating a Retrieval-Augmented Generation (RAG) system into engineering tools. The application demonstrated the potential of language models to assist with labor-intensive tasks such as specification writing, code generation, testing, and documentation. While current solutions show promise, their maturity varies significantly due to limitations in curated training data and concerns around data privacy in commercial models. The authors suggested that future AI assistants in automation engineering will likely rely on hybrid architectures combining different language models based on task type and data sensitivity. They also highlighted the need to refine further embedding models in RAG systems and multimodal approaches for vision-related automation tasks [15].

Zheng et al. tackled the challenge of using LLMs to generate ST code. As the diversity of vendor-specific platforms complicates the adaptation process, they created the PLC-100 dataset containing structured prompt examples (10 types of 100 prompt words) and developed a GPT-4-based generation approach paired with an adaptive evaluation system. Their experiments showed that the dataset and evaluation scheme improved the reliability and usability of generated PLC code, although limitations remain in handling domain-specific complexity. The authors plan to refine evaluation benchmarks and adaptive indicators to better assess code quality, while also exploring the broader opportunities and constraints of LLMs in industrial automation in the future [18].

3.3.2. Domain-specific language models

Ensuring reliability and verification within industrial automation systems is critical for maintaining operational integrity and safety. This subsection explores the contributions of pretrained language models to these essential areas, offering robust solutions for code verification and generating graphical programming languages. By addressing these challenges, the studies presented herein advance the discourse on secure and efficient automation processes.

Haag et al. addressed the difficulties of limited training data and the syntactic complexity of ST programming by proposing a framework for fine-tuning LLMs to generate more reliable ST code. Their approach combines supervised fine-tuning with Direct Preference Optimisation (DPO) and an iterative, preference-based training cycle. A central element of the method is the pairing of compiler-based syntactic validation with feedback from an LLM-driven semantic evaluator. Although this evaluator is not optimised for actual code generation, it plays a crucial role in checking functional correctness. The system operates by iteratively generating and testing synthetic training samples, allowing gradual performance improvements. The advantages of the setup are that no heavy manual annotation is needed to enhance performance. The study shows increased compilation success rate and semantic accuracy. The authors point toward specialised industrial copilots for PLC programming to ease the issue of domain-specific data scarcity and enable a more cost-effective alignment with established industrial coding standards [19].

Fakih et al. introduced the LLM4PLC framework, a structured, user-guided iterative pipeline designed to enhance the reliability and verifiability of code automatically generated by LLMs for Industrial Control Systems (ICS) that utilise PLCs. The authors recognised the limitations of current LLMs, such as GPT-4 and LLaMa2, in producing syntactically and semantically valid PLC code, such as insufficient support for domain-specific languages and the absence of formal correctness guarantees. The LLM4PLC framework works by combining user feedback with external validation mechanisms, for example grammar checkers, compilers, and Symbolic Model Verification (SMV) tools. The framework is further enhanced through Prompt Engineering strategies and lightweight model adaptation using Low-Rank Adaptation (LoRA) fine-tuning. After validation on a FischerTechnik Manufacturing TestBed (MFTB), results showed that LLM4PLC significantly improves code generation. Success rates increased from 47% to 72% and expert-rated code quality scores elevated from 2.25 out of 10 to 7.75 out of 10. These findings show the potential of LLM4PLC to be a link between general-purpose language models and the specific requirements of ICS programming [20].

Liu et al. introduced Agents4PLC, a pioneering multi-agent framework leveraging LLMs for the automated generation and formal verification of PLC code in industrial control systems. The framework bridges the gap between natural language specifications and functionally correct, verifiable control logic. The authors established a closed-loop PLC code synthesis foundation by constructing a comprehensive benchmark to transition from human-made requirements to verified and validated PLC code specifications. Agents4PLC integrates several methodologies from Retrieval-Augmented Generation (RAG) to chain-of-thought prompting and specialised prompt engineering to enhance LLMs' contextual reasoning and domain-specific applicability. The system's modular structure incorporates various base code-generation models while emphasising code-level verification. Empirical evaluations showed that Agents4PLC has high automation fidelity and is robust in verification. Results demonstrated that it outperformed existing approaches by a large margin across meticulous performance metrics, highlighting the potential of LLM-based agent systems in automation. The authors aim to extend language and standards compatibility and introduce user-centred feedback mechanisms to iteratively refine agent behaviour and improve real-world deployment in the future [16].

Koziolek et al. introduced an LLM-based approach for automatically generating test cases for control logic in PLCs and DCS to address the limitations of traditional testing methodologies. Conventional strategies, such as symbolic execution and search-based techniques, are effective in identifying specific programming faults but often require formal specifications. They can also suffer from state space explosion, especially when dealing with informal or loosely defined logic. The proposed solution leverages an LLM to synthesise test inputs and outputs using direct code prompts to program Function Blocks. After validating on ten publicly available function blocks from the OSCAT library, results demonstrated the usability and efficiency of this model, achieving high statement coverage for low- to medium-complexity programs. The authors developed an enhanced prompting strategy to address a common but significant drawback in LLM reasoning, often leading to incorrect or illogical assertions, requiring manual post-processing by human experts. However, even with this setup, support for complex elements, such as timers, remained limited. The research suggests that LLM-based test generation could be integrated into industrial development environments to reduce manual effort. Moreover, the authors proposed future extensions, such as agent-based orchestration, chain-of-thought prompting, and hybrid approach with symbolic execution and search-based testing to enhance the quality and reliability of the generated test suites [23].

Zhang et al. turned their attention to the often-overlooked area of graphical programming languages, which has received less focus compared to the dominance of ST in AI-assisted PLC programming. Their study looked at how LLMs might help generate graphical forms, focusing on SFC and LD. Experiments using prompt engineering and example-driven inputs showed that LLMs can effectively generate simple SFCs. Results also demonstrated that the same models face difficulties with LD generations, particularly with tasks involving logic comprehension and interconversion. While basic

state machine behaviour (e.g., 1-bit and 2-bit logic) was handled well in SFC form, more complex logic and automatic LD–SFC transformation were unreliable. The authors noted the importance of gaining theoretical insights, as current evaluations are based entirely on black-box methods. Proposed future enhancements include, for example, fine-tuning, Retrieval-Augmented Generation (RAG), and improving ASCII space handling to enhance output accuracy [24].

Salari et al. addressed the limited applicability of automated test generation in PLC software development, particularly for safety-critical systems, by introducing PyLC. The developed framework converts PLC programs (written in FBD and ST) into Python code while applying testing and automated test generation techniques on the Python-based unit and connecting industrial environments such as CODESYS with contemporary testing frameworks. PyLC was validated using behaviour-based, rule-based, and coverage-based testing methods on real industrial programs, showing promise in improving testability and regression testing support. While the implementation still requires manual steps, the authors plan to enhance PyLC with full automation, integration with CODESYS Test Manager, and formal verification features. Future work also includes exploring the efficiency of different search-based algorithms to generate high-quality test cases for evolving control software [29].

Ren et al. addressed the lack of open-source PLC code datasets and the difficulty of using LLMs to generate PLC programs. They built *ST4Indux*, a dataset of ST programs for automation tasks, and introduced the Control Logic-Guided Iterative Fine-Tuning (CLIFT) method to enhance model performance systematically. Based on these, they developed *MetaIndux-PLC*, an LLM specialised in PLC code generation, which demonstrated improved accuracy in producing complex motion control code and better compliance with industrial requirements. This approach reduced programming effort and costs while increasing flexibility. However, the solution is still limited by the scale of available training data and constrained by the inherent complexity of industrial systems. The authors noted the need for further advances in dataset coverage, validation techniques, and long-term reliability [17].

4. Application and case studies

This section presents the previously introduced articles and sorts them according to their PLC language focus. By analyzing these applications, these examples can provide insights into challenges and solutions that are yet to appear in industrial automation. From adaptive manufacturing systems to graph-based control languages, these case studies emphasize the potential of AI-based automation technologies.

4.1. Textual programming languages

LLMs have been applied to generate and verify ST programs in various experimental and semi-industrial contexts. Koziolek et al. [8] demonstrated that GPT-4 can generate syntactically correct ST code from prompts written in the human language. This demonstrates the feasibility of the approach but lacks runtime validation. In subsequent research [9], the authors performed benchmarks on various LLMs using representative ST tasks, identifying performance limitations with logic-heavy prompts in the process. Fan Mo et al. [27] introduced an ontology-guided GNN framework that generated adaptive ST code from device configurations, which was validated using real industrial IoT datasets. Haag et al. [19] applied discrete program optimization for code synthesis from I/O traces and function block specifications, targeting vendor-specific constraints. Fakih et al. [20] proposed a verification pipeline to evaluate LLM-generated ST, using a corpus of tasks to assess robustness, although real-time execution tests were not included. Tran et al. [12] focused on a PLC edge platform capable of learning from distributed data and generating ST control logic in a simulated smart manufacturing testbed. Liu et al. [16] used multi-agent AI planning to compose complete ST programs from task descriptions, showing end-to-end capability in predefined industrial scenarios. Salari et al. [29] introduced PyLC, a Python-based ST/FBD testing toolkit that facilitates functional test conversion and debugging but was validated in a controlled environment. Xia et al. [11] developed a predictive tool to identify potential faults in vendor-specific ST logic using domain-trained classifiers. Ren et al. [17] created the ST4Indux dataset and the CLIFT fine-tuning method to improve LLMs for ST programming and developed MetaIndux-PLC to reduce programming effort and costs. Zheng et al. [18] created a PLC-100 dataset of structured prompts and developed a GPT-4-based method to generate PLC code with an adaptive evaluation system. They demonstrated that this approach improves the reliability and operability of generated code while highlighting remaining challenges in domain-specific adaptability. Together, these works demonstrate growing maturity in LLM-based and hybrid ST automation, with promising toolchains that remain largely in prototype or offline evaluation stages.

4.2. Graphical programming languages

Research on automating graphical PLC languages, such as FBD, SFC, and LD, has focused on logic synthesis and testing support. Cai et al. [7] introduced MODI, a rule-based generator for FBD code tailored to building automation domains, successfully applied to HVAC and lighting use cases using real configuration files. Ogundare et al. [21] presented a recurrent neural network–based no-code interface for FBD generation from textual task specifications; their method was tested on manually prepared datasets, though not deployed in live systems. Liu et al. [22] proposed MuFBDTester,

which applies mutation testing to enhance the validation of existing FBD programs, showing higher fault detection in structured test environments. Koziolek et al. [23] extended LLM applications to FB test generation using input/output specifications; while syntactically sound outputs were achieved, runtime verification remains future work. Zhang et al. [24] evaluated SFC and LD code generation via structured prompt engineering, producing plausible logic structures validated by expert review. Löppenberg et al. [25] applied a grammar-based evolutionary algorithm to synthesise correct SFC/LD implementations from desired behaviour specifications, focusing on logic coverage and compactness. Sidorenko et al. [28] proposed using Behaviour Trees as a semantic abstraction for IEC 61499 and IEC 61131-3 environments, enabling visually guided, modular control strategies evaluated in distributed systems. These studies show active research and progress in graphical code automation within the past 5 years and demonstrate potential use in safety-focused fields, though often in testbeds or simulated environments.

5. Challenges, limitations, and future directions

Despite their impressive performance, many challenges remain with AI-based industrial automation. Many of these difficulties are rooted in the well-known lack of explainability and sensitivity to disturbances of current AI systems. For example, verifying the correctness of generated code, maintaining context-awareness in complex or layered control tasks, and ensuring compliance with vendor-specific requirements are still open problems. On top of this, hallucinations or simple misinterpretations of informal prompts can lead to faulty or misleading outputs. Several approaches – such as fine-tuning, iterative validation cycles, or retrieval-augmented generation – offer partial remedies but rarely solve the problem completely. There will be increasing concerns about cybersecurity once these systems are made accessible to external users.

The research is focusing hybrid solutions and applications that combine LLM-based agents with formal verification methods and domain-specific fine-tuning. Stronger integration into existing industrial tool chains and simulation platforms is also expected, together with closed-loop validation and real-time feedback. Another area with considerable potential is support for graphical IEC 61131-3 languages, especially SFC and FBD, which have received relatively little attention. If these directions mature, LLMs could become helpful assistants and active collaborators in control system development, helping build more adaptive and efficient automation workflows.

6. Discussion

This review has outlined how AI, especially LLMs are beginning to shape the automation of PLC programming. Across the literature, recurring trends have been identified: high levels of syntactic validity in generated ST code, an increasing degree of semantic accuracy, and the ability to reduce development time quite significantly. The often-cited benefits include easier access for non-expert users, better modularity and scalability through prompt engineering or ontology-based approaches, and support for test automation and natural language specifications. The next milestone to achieve is creating a copilot made by integrating AI models, each with an exceptional understanding of a specific PLC language, until all languages are accounted for. In summary, the research findings indicate a promising potential for fully automating PLC programming in the near future.

Acknowledgements

The project was supported by the Doctoral Excellence Fellowship Program (DCEP), and 2021-1.2.4-TÉT-2021-00054 „Machine learning supported process monitoring of micro machining in the frame of Industry 4.0” funded by the National Research Development and Innovation Fund of the Ministry of Culture and Innovation and the Budapest University of Technology and Economics. Additionally, this research was partially funded by the János Bolyai Research Scholarship from the Hungarian Academy of Sciences BO/00841/24/6 and by the National Research, Development and Innovation Office, project number OTKA SNN 146940, entitled “Basic Investigation of the Applicability of Artificial Intelligence Based Predictive Models to Improve the Quality of Production with Advanced Machining Processes”.

REFERENCES

[1] Siemens and Microsoft scale industrial AI, Microsoft Source, Oct. 24, 2024 https://news.microsoft.com/source/2024/10/24/siemens-and-microsoft-scale-industrial-ai/ (accessed: Jul. 03, 2025)

[2] Rockwell Automation and Microsoft deliver on a shared vision to accelerate industrial transformation, Rockwell Automation, UK https://www.rockwellautomation.com/en-gb/company/news/press-releases/Rockwell-Automation-and-Microsoft-Deliver-on-a-Shared-Vision-to-Accelerate-Industrial-Transformation.html (accessed: Jul. 03, 2025)

[3] Rockwell Automation and Microsoft expand partnership to simplify industrial transformation, Microsoft Source, Oct. 06, 2020 https://news.microsoft.com/source/2020/10/06/rockwell-automation-and-microsoft-expand-partnership-to-simplify-industrial-transformation/ (accessed: Jul. 03, 2025)

[4] Schneider Electric charges into the future with Microsoft Copilot for Sales, Microsoft Customer Stories https://www.microsoft.com/en/customers/story/18860-schneider-electric-microsoft-365-copilot-for-sales (accessed: Jul. 03, 2025)

[5] Schneider industrial generative AI copilot https://www.processexcellencenetwork.com/ai/news/schneider-electric-launches-industrial-generative-ai-copilot-with-microsoft (accessed: Jul. 03, 2025)

[6] Hu, X.; Liang, Y.; Zhu, S.; Li, H.; Zhu, S.: ST-Petri: A visual executable semantic model for PLC structured text language, *Proc. 2024 IEEE 22nd Int. Conf. Ind. Inform. (INDIN), Beijing, China*, 2024, 1–6, DOI: 10.1109/INDIN58382.2024.10774532

[7] Cai, X.; Shi, R.; Kümpell, A.; Müller, D.: Automated generation of PLC code for implementing mode-based control algorithms in buildings, *Proc. 2022 30th Mediterr. Conf. Control Autom. (MED), Vouliagmnei, Greece*, 2022, 1087–1092, DOI: 10.1109/MED54222.2022.9837182

[8] Koziolek, H.; Gruener, S.; Ashiwal, V.: ChatGPT for PLC/DCS control logic generation, 2023, *arXiv*, arXiv:2305.15809, DOI: 10.48550/arXiv.2305.15809

[9] Koziolek, H.; Grüner, S.; Hark, R.; Ashiwal, V.; Linsbauer, S.; Eskandani, N.: LLM-based and retrieval-augmented control code generation, in: Proceedings of the 1st International Workshop on Large Language Models for Code (LLM4Code '24) (Association for Computing Machinery, New York, NY, USA), 2024, pp. 22–29, DOI: 10.1145/3643795.3648384

[10] Yang, D.; Wu, A.; Zhang, T.; Zhang, L.; Liu, F.; Lian, X.; Ren, Y; Tian, J.: A multi-agent framework for extensible structured text generation in PLCs, Dec. 03, 2024, *arXiv*, arXiv:2412.02410, https://arxiv.org/abs/2412.02410v1

[11] Xia, W.; Zhang, Y.; Zhao, B.; Liu, W.; Han, L.; Ye, Q.: Intelligent PLC code generation in HCPS 2.0: A multi-dimensional taxonomy and evolutionary framework, in Proceedings of the 2025 2nd International Conference on Generative Artificial Intelligence and Information Security (Association for Computing Machinery, New York, NY, USA), 2025, pp. 202–212, DOI: 10.1145/3728725.3728757

[12] Tran, K.; Zhang, J.; Pfeiffer, J.; Wortmann, A.; Wiesmayr, B.: Generating PLC code with universal Large Language Models, *Proc. 2024 IEEE 29th Int. Conf. Emerg. Technol. Fact. Autom. (ETFA), Padova, Italy*, 2024, 1–8, DOI: 10.1109/ETFA61755.2024.10711113

[13] Boudribila, A.; Chadi, M.-A.; Tajer, A.; Boulghasoul, Z.: Large Language Models and adversarial Reinforcement Learning to automate PLCs programming: A preliminary investigation, *2023 9th Int. Conf. Control Decis. Inf. Technol. (CoDIT) Rome, Italy*, 2023, 650–655, DOI: 10.1109/CoDIT58514.2023.10284185

[14] Koziolek, H.; Koziolek, A.: LLM-based control code generation using Image Recognition, in Proceedings of the 1st International Workshop on Large Language Models for Code (LLM4Code '24) (Association for Computing Machinery, New York, NY, USA), 2024, pp. 38–45, DOI: 10.1145/3643795.3648385

[15] Reinhardt, D.; Jeschin, J.; Jasperneite, J.; Benndorf, G.: ChatPLC – Potenziale der Generativen KI für die Steuerungsentwicklung, *Z. Wirtsch. Fabrikbetr.*, 2025, **120**(s1), 196–201, DOI: 10.1515/zwf-2024-0121

[16] Liu, Z.; Zeng, R.; Eang, D.; Peng, G.; Wang, J.; Liu, Q.; Liu, P.; Wang, W.: Agents4PLC: Automating closed-loop PLC code generation and verification in industrial control systems using LLM-based agents, Dec. 25, 2024, *arXiv*, arXiv:2410.14209, DOI: 10.48550/arXiv.2410.14209

[17] Ren, L.; Wang, H.; Dong, J.; Wang, H.; Liu, S.; Laili, Y.; Zhang, L.: MetaIndux-PLC: A Control Logic-Guided LLM for PLC code generation in industrial control systems, *Appl. Soft Comput.*, 2025, **184**, 113673, DOI: 10.1016/j.asoc.2025.113673

[18] Zheng, J.; Li, D.; Sun, Y.; Song, C.: A novel method of PLC code generation based on Large Language Models, *Proc. 2025 6th Int. Conf. Comput. Eng. Appl. (ICCEA), Hangzhou, China*, 2025, 1440–1444, DOI: 10.1109/ICCEA65460.2025.11102210

[19] Haag, A.; Fuchs, B.; Kacan, A.; Lohse, O.: Training LLMs for generating IEC 61131-3 structured text with online feedback, 2024, *arXiv*, arXiv:2410.22159, DOI: 10.48550/arXiv.2410.22159

[20] Fakih, M.; Dharmaji, R.; Moghaddas, Y.; Quiros, G.; Ogundare, O.; Al Faruque, M.A.: LLM4PLC: Harnessing Large Language Models for verifiable programming of PLCs in industrial control system, in Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '24) (Association for Computing Machinery, New York, NY, USA), 2024, pp. 192–203. DOI: 10.1145/3639477.3639743

[21] Ogundare, O.; Araya, G.Q.; Qamsane, Y.: No Code AI: Automatic generation of Function Block Diagrams from documentation and associated heuristic for context-aware ML algorithm training, 2023, *arXiv*, arXiv:2304.04117, DOI: 10.48550/arXiv.2304.04117

[22] Liu, L.: Automated mutation-adequate test generation for Function Block Diagram programs (PhD thesis), Korea Advanced Institute of Science and Technology, 2021 https://koasas.kaist.ac.kr/handle/10203/296121 (accessed: Jun. 24, 2025)

[23] Koziolek, H.; Ashiwal, V.; Bandyopadhyay, S.; Chandrika, K.R: Automated control logic test case generation using Large Language Models, *Proc. 2024 IEEE 29th Int. Conf. Emerg. Technol. Fact. Autom. (ETFA), Padova, Italy*, 2024, 1–8, DOI: 10.1109/ETFA61755.2024.10711016

[24] Zhang, Y.; de Sousa, M.: Exploring LLM support for generating IEC 61131-3 graphic language programs, *Proc. 2024 IEEE 22nd Int. Conf. Ind. Inform. (INDIN), Beijing, China*, 2024, 1–7, DOI: 10.1109/INDIN58382.2024.10774464

[25] Löppenberg, M.; Schwung, A.: Structured graph generation by evolutionary algorithm for program code development, *IECON 2024 - 50th Ann. Conf. IEEE Ind. Electron. Soc., Chicago, IL, USA*, 2024, 1–6. DOI: 10.1109/IECON55916.2024.10905556

[26] Löppenberg, M.; Schwung, A.: Self optimisation and automatic code generation by evolutionary algorithms in PLC based controlling processes, *2023 IEEE 21st International Conference on Industrial Informatics (INDIN), Lemgo, Germany,* 2023, 1–6, DOI: 10.1109/INDIN51400.2023.10218168

[27] Mo, F.; Querejeta, M.U.; Hellewell, J.; Rehman, H.U.; Illarramendi,Rezabal, M.; Chaplin, J.C.; Sanderson, D.; Ratchev, S.: PLC orchestration automation to enhance human–machine integration in adaptive manufacturing systems*, J. Manuf. Syst.*, 2023, **71**, 172–187, DOI: 10.1016/j.jmsy.2023.07.015

[28] Sidorenko, A.; Rezapour, M.; Wagner, A.; Ruskowski, M.: Towards using behavior trees in industrial automation controllers, 2024, *arXiv*, arXiv:2404.14030, DOI: 10.48550/arXiv.2404.14030

[29] Ebrahimi Salari, M.; Enoiu, E.P.; Afzal, W.; Seceleanu, C.: PyLC: A framework for transforming and validating PLC software using Python and Pynguin test generator, in Proceedongs of the 38th ACM/SIGAPP Symposium on Applied Computing (SAC '23) (Association for Computing Machinery, New York, NY, USA), 2023, pp. 1476–1485. DOI: 10.1145/3555776.3577698

[30] Brecher, C.; Weck, M.: Machine tools production systems 3 - Mechatronic systems, control and automation (Springer), 2022, DOI: 10.1007/978-3-658-34622-5

[31] IEC 61131-3:2025 Programmable controllers – Part 3: Programming languages, International Electrotechnical Commission, 2025 https://webstore.iec.ch/en/publication/68533