Benchmarking Agentic AI Architectures for Automation Engineering

Ilona Bühlmann
Institute of Control Systems
Karlsruhe Institute of Technology
Karlsruhe, Germany
ORCID: 0009-0007-6163-011X

Marwin Madsen
Institute of Control Systems
Karlsruhe Institute of Technology
Karlsruhe, Germany
ORCID: 0009-0006-9953-2382

Christian Späth
Global Automation, Strategy & Technology
Boehringer Ingelheim
Biberach, Germany
ORCID: 0009-0005-8675-0068

Fabian Pfetzinger
Institute of Control Systems
Karlsruhe Institute of Technology
Karlsruhe, Germany
fabian.pfetzinger@student.kit.edu

Frank Maurer
Global Automation, Strategy & Technology
Boehringer Ingelheim
Biberach, Germany
frank.maurer@boehringer-ingelheim.com

Mike Barth
Institute of Control Systems
Karlsruhe Institute of Technology
Karlsruhe, Germany
ORCID: 0000-0003-2337-063X

***Abstract*—Developing automation software requires not only control-logic generation but also hardware configuration, I/O mapping, and interaction with vendor-specific engineering tools. Despite the success of large language models (LLMs) and agentic AI in conventional software engineering, their utility for end-to-end automation engineering is under-explored. This paper compares seven architectures, ranging from a manual LLM baseline to standardized tool-enabled single-agent and multi-agent systems with optional retrieval-augmented generation and inter-agent communication. The benchmark is evaluated on two industrial use cases and measures syntactic correctness and semantic correctness against predefined hardware test cases, tool calls, token consumption, and cost. The results show that explicit orchestration and in particular access to the vendor-specific engineering tool substantially improves functional performance over plain LLM baselines. Single-agent setups using vendor-specific tools and closed-loop orchestration performed best. Multi-agent and RAG enhancements provided negligible gains, likely because the current task complexity does not justify their additional coordination overhead. These findings indicate that lightweight agent designs are currently a practical choice for industrial automation engineering, while standardized tool and inter-agent protocols remain promising for modularity and extensibility.**

***Index Terms*—IEC 61131-3, Agentic AI, Benchmarking, code generation, tool integration, MCP, A2A Communication**

I. Introduction

Programmable logic controllers (PLCs) are central to industrial automation, with workflows spanning control logic, hardware configuration, and data mapping across heterogeneous vendor tools. Despite increasing standardization in engineering practice, the implementation of automation projects is still largely manual, making the process time-consuming and error-prone. Particularly, hardware-related tasks such as network configuration and I/O mapping are well-known bottlenecks in automation engineering [1]. Prior studies estimate that 50% to 60% of total engineering effort is software-related, with engineers using multiple vendor-specific tools to implement control programs and configure hardware systems [2].

Recent advances in large language models (LLMs) and in agentic artificial intelligence (AI) have improved productivity in software engineering, especially for code generation and workflow automation. Multi-agent systems have attracted attention because they can decompose complex tasks into specialized roles, potentially improving task coverage, modularity, and output quality [3], [4]. Conversely, studies in software engineering show that simpler single-agent architectures can match or surpass multi-agent systems when considering setup complexity, coordination overhead, latency, and token consumption [5]. These trade-offs are relevant in automation, where tasks extend beyond source-code generation.

Initial research on AI-assisted PLC code generation for industrial automation has shown promising results. LLM4PLC [6] demonstrated that LLM-based, human-in-the-loop workflow augmented with compiler feedback and model checking improves PLC code generation success and expert-rated quality via prompt engineering and fine-tuning. Agents4PLC [7] extended this line of work with a closed-loop multi-agent architecture. It combines Retrieval-Augmented Generation (RAG) [8], i.e. augmenting the LLM context with a knowledge base, and structured prompting to automate the path from natural-language requirements to verified PLC code, strengthening functional assurance.

AutoPLC [9] and Vendor-Aware Industrial Agents [10] highlighted that vendor-aware conditioning and RAG-grounded documentation reduce compilation mismatches and hallucinations which is especially important for Structured-Text (ST) dialects.

However, despite this rapid progress, the existing literature still lacks a reproducible benchmark that compares different agentic AI architectures under a common evaluation frame-

© 2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

work. In particular, no prior study provides a cross-architecture evaluation that simultaneously

(i) compares the spectrum from baseline and single-agent to role-specialized multi-agent systems,
(ii) evaluates tool integration in the automation domain by using standardized and open-source based approaches (e.g. Model Context Protocol (MCP) [11]),
(iii) investigates interoperability across heterogeneous role-specialized agents in multi-agent systems through standardized and open-source based approaches (e.g. Agent-to-Agent (A2A) [12] protocol), and
(iv) evaluates both functional and non-functional metrics, including compilation success, functional correctness, token consumption, tool calls (e.g. API utilization), and time-to-solution.

To address this gap, this paper presents a benchmark suite for the systematic evaluation of seven progressively more complex agentic AI architectures in two real-life industrial use cases. Beyond comparing architectural performance, the paper aims to assess the feasibility of end-to-end agentic AI-based automation engineering, including the IEC 61131-3 code generation, I/O mapping, hardware and network configuration within a compilable project structure. Furthermore, the role of standardized protocols in the context of agentic AI is examined for integration with the automation toolchain.

II. Architectural Fundamentals and Related Work

This section reviews recent approaches to PLC code generation based on LLMs and agentic AI. Recent work on PLC code generation with LLMs and agentic AI has surged since ChatGPT's 2022 release [20], bringing generative AI into automation engineering. The literature research focuses on papers (i) published between 2022-2026, (ii) focusing primarily on PLC development/integration using LLM- or agentic-AI methods, and (iii) providing enough methodological detail to compare architectures, knowledge sources, tooling, and evaluation criteria.

A substantial part of the early literature consists of exploratory LLM-based approaches. Koziolek et al. investigated the use of off-the-shelf LLM to generate PLC code from different engineering artifacts and data formats along the automation life cycle. Their studies [13]–[15], [18] show that general-purpose LLMs can generate syntactically plausible IEC 61131-3 structured text, but they also highlight the need for stronger grounding and validation. In a later study [15], the same line of work demonstrated that RAG is beneficial for retrieving knowledge from vendor-specific development environments and facilitates the use of proprietary function blocks in generated PLC code. These contributions are important in establishing the feasibility of LLM-assisted PLC programming, but they do not provide a systematic comparison across architectural paradigms.

Ren et al. introduced MetaIndux-PLC [17], a specialized LLM for generating ST Code for complex motion-control tasks. Besides ST, research has also examined the generation of graphical IEC 61131-3 languages such as Sequential Function Chart (SFC) and Ladder Diagram [16]. These results show that LLMs can handle SFC reasonably well, whereas generating Ladder Diagrams from textual descriptions remains considerably more challenging.

More structured workflows have been proposed in subsequent work. The authors of LLM4PLC [6] introduced a human-in-the-loop workflow that integrates user feedback, prompt engineering, model checking, and compiler feedback into an automated engineering loop. To improve their results, they used Low-Rank Adaptation, a technique that adapts AI models to new contexts by adding lightweight trainable components rather than retraining the entire model. While this improves model performance with lower memory and computational demands, the approach still requires curated datasets and remains focused primarily on PLC code generation rather than broader automation-engineering integration.

A different line of research emphasizes vendor awareness. AutoPLC [9] showed that conditioning generation on vendor-specific libraries and tools can reduce compilation mismatches across PLC manufacturers and thereby improve deployability in heterogeneous toolchains. The study also highlights differences among the ST dialects of CODESYS, Beckhoff TwinCAT, and Siemens SCL, underscoring vendor-specific syntax as a major source of compilation mismatch.

Kersting et al. [10] extended this idea with a vendor-aware single-agent system that combines compiler-in-the-loop generation and a competitive model layer in which different models generate code against each other. This work is grounded in domain-specific knowledge sources for Mitsubishi MELSOFT GX Works3. More broadly, these studies underline that PLC code generation cannot be treated as a purely syntax-oriented problem. Practical success depends on access to vendor-specific constraints, libraries, and engineering conventions.

Similarly, recent dual-knowledge and graph-enhanced multi-agent approaches further suggest that specialized coordination mechanisms can improve syntactic and functional consistency [19]. At the same time, the broader agentic AI literature raises whether these gains justify the additional coordination overhead, especially in engineering workflows where token consumption, latency, and tool interaction costs matter [5].

Multi-agent approaches represent the most explicit attempt to structure PLC code generation into specialized subtasks. Agents4PLC [7] structures the workflow into role-specialized agents for retrieval, planning, coding, debugging, and validation. The study illustrates the benefits of closed-loop verification and ablation-based analysis. Its combination of RAG, syntax hints, and Chain-of-Thought prompting increases verifiable and compilation-pass rates for more complex tasks, but it also raises the question of when the additional coordination overhead is justified by measurable performance gains.

A comprehensive summary of the introduced approaches is summarized in Table I. In summary, the literature proposes a variety of approaches for PLC code generation in automation engineering, but it also shows several important limitations. First, most existing studies focus on isolated code-generation

Table I: Comparison Table: In category *Tools* the ○ describes the use of open-source tools and ● the use of proprietary tools.

Identification			Knowledge Source			Tools	E/E	Validation		
Author	Year	Publication	Prompt	Training	RAG			Syntactic	Semantic	Efficiency
LLM-based approaches										
Koziolek et al. [13]	May, 2023	ChatGPT for PLC/DCS	✓	×	×	○	×	×	×	×
Koziolek et al. [14]	Nov, 2023	Image Recognition	✓	×	×	○	×	×	×	×
Fakih et al. [6]	Jan, 2024	LLM4PLC	×	✓	×	○	×	✓	×	✓
Koziolek et al. [15]	Apr, 2024	RAG	×	×	✓	○	×	×	×	×
Zhang et al. [16]	Oct, 2024	Graphical IEC 61131-3	×	✓	×	○	×	✓	✓	×
Ren et al. [17]	Jul, 2025	MetaIndux-PLC	×	✓	×	○	×	✓	✓	✓
Yang et al. [9]	Aug, 2025	AutoPLC	×	×	✓	●	×	✓	×	×
Koziolek et al. [18]	Oct, 2025	Spec2Control	✓	×	×	○	×	✓	✓	✓
		Architecture A0	✓	×	×	●	×	✓	✓	✓
		Architecture A1	✓	×	×	●	✓	✓	✓	✓
Single-Agent approaches										
Kersting et al. [10]	Jan, 2026	Vendor-Aware Industrial Agents	×	×	✓	●	×	✓	×	×
		Architecture A2	✓	×	×	●	✓	✓	✓	✓
Multi-Agent approaches										
Liu et al. [7]	Dec, 2024	Agents4PLC	×	×	✓	○	×	✓	✓	×
Zhang et al. [19]	Jan, 2026	MPC-Coder	×	×	✓	○	×	✓	✓	×
		Architecture A3	✓	×	×	●	✓	✓	✓	✓
		Architecture A4	×	×	✓	●	✓	✓	✓	✓
		Architecture A5	✓	×	×	●	✓	✓	✓	✓
		Architecture A6	×	×	✓	●	✓	✓	✓	✓

tasks rather than complete automation-engineering workflows. Second, the majority of published approaches are either monolithic LLM workflows or narrowly defined specialized systems, while direct comparisons between LLM-based, single-agent, and multi-agent architectures remain scarce. Third, validation practices are heterogeneous: most studies emphasize syntactic correctness, compilation success, or expert judgment, whereas broader non-functional metrics such as token consumption, tool usage, and engineering effort are reported only inconsistently. Finally, although some studies incorporate vendor-specific knowledge, full end-to-end integration of PLC code, I/O mapping, hardware, and network configuration remains largely unaddressed.

For a structured comparison, the literature is analyzed along five aspects: *architecture*, *knowledge source*, *tooling*, *end-to-end integration*, and *validation*. Architecture and knowledge source cluster each work into LLM-based, single-agent, or multi-agent systems and compare how domain knowledge is supplied. An aspect that strongly affects code quality given vendor-specific tools and dialects [9].

For classifying the architectures proposed in the literature, the definition of agentic AI according to [21] is applied. **LLM-based approaches** are architectures that employ a large language model as the core code-generation engine and rely on a Human-in-the-Loop feedback mechanism to correct and refine the generated output.

Single-Agent systems consist of a single LLM-agent that receives an initial task prompt from the control engineer. After the first generation step the engineer is no longer involved and the agent produces the subsequent correction prompts and iteratively refines the code until the desired quality is reached.

Multi-Agent systems extend the single-agent concept by deploying several specialized agents (e.g., retrieval, planning, coding, debugging, verification). The control engineer also supplies the initial task description. Agents exchange feedback among themselves (e.g., via an A2A protocol), eliminating further human intervention.

The dimensions *tooling* and *end-to-end integration* characterize the implementation level of the respective approaches. More specifically, the review evaluates whether the workflow rely on open-source tools or on proprietary tools. Open-source toolchains improve reproducibility in scientific settings, whereas proprietary tools better reflect industrial practice and expose challenges related to vendor lock-in. In addition, *end-to-end integration* assesses whether an approach generates only PLC code or produces a complete and deployable automation-engineering project, including configuration files, hardware mapping and a buildable project structure that can be compiled and tested with minimal manual effort.

Finally, the *validation* category describes the quantitative and qualitative metrics reported for each approach. A consistent comparison of validation criteria is important because existing studies differ substantially in whether they report syntax correctness and semantic correctness. Further it is evaluated whether the suggested works provide a benchmarking on resource efficiency such as token efficiency, costs or tool calls.

The benchmark presented in this paper responds to these limitations by comparing seven architectures within a common industry-oriented setting, by evaluating the role of standardized tool and communication layers through MCP and A2A, and by jointly considering functional and non-functional metrics relevant to end-to-end automation engineering.

III. Agentic AI Architectures and Concepts

This section outlines the software stack and the architectures evaluated in the benchmark. The benchmark progresses from a manual LLM baseline to standardized multi-agent systems, illustrating increasingly structured agentic AI workflows and relies on two standardization mechanisms:

- MCP: A machine-readable interface that unifies request/response formats across the tool chain, separating orchestration from the underlying tools and enabling easy reuse of engineering services [11].
- A2A: A protocol that standardizes communication among heterogeneous agents, allowing to test whether a common inter-agent layer reduces integration effort, mitigates vendor lock-in and improves modularity in multi-agent automation workflows [12].

A. LLM-based Architecture (A0 - A1)

The first two architectures establish the benchmark baseline for the subsequent agent-based workflows and are depicted in Figure 1.

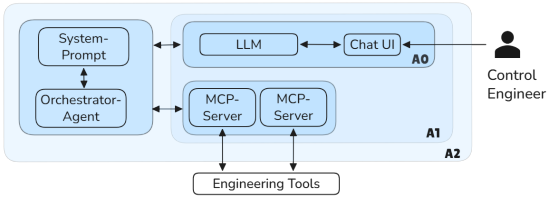


Figure 1: LLM baseline and Single-Agent Architecture (A2)

Architecture **A0** represents a human-in-the-loop chat interface with a plain LLM instead of a specialized agent workflow or direct tool integration. It serves as the reference point for estimating accuracy and effort when PLC projects are generated ad hoc and therefore rely primarily on the prompt and the model capabilities. This baseline enables a structured comparison and remains relevant for assessing at which level of task complexity additional context, tool access, and orchestration become necessary. Architecture **A1** extends this by integrating two MCP servers for I/O and PLC-code functions (e.g., upload/compilation) to evaluate if standardized tool access improves accuracy, despite the potential for increased prompt complexity [22].

B. Single-Agent Architecture (A2)

Architecture A2 introduces a single-agent closed-loop workflow (Fig. 1), moving orchestration from user interaction to a persistent system prompt. By incorporating domain knowledge and correction strategies, A2 iteratively refines outputs until a goal is reached without further human prompting. This architecture evaluates whether a single prompt-grounded agent can match the performance of multi-agent systems while eliminating their associated coordination overhead.

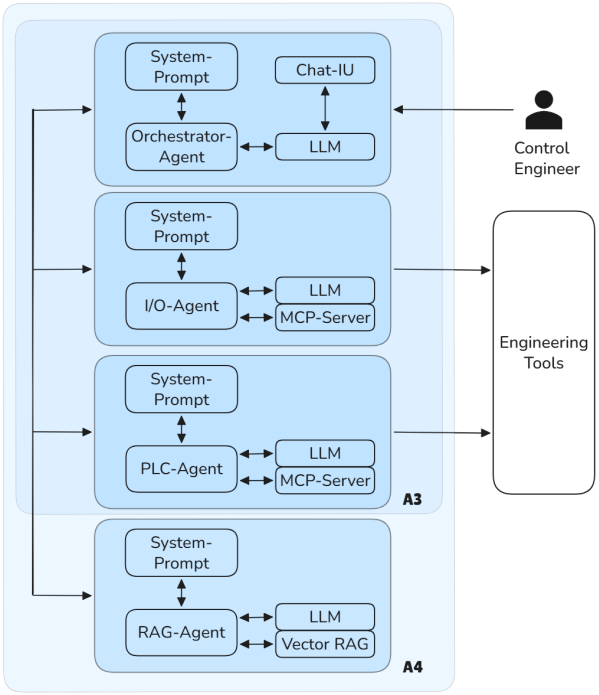


Figure 2: Multi-Agent Engineering Workflow

C. Multi-Agent Architecture (A3 - A6)

Architectures **A3 to A6** extend the workflow by introducing role-specialized agents, as illustrated in Fig. 2.

The overall task is decomposed into the dedicated agents for I/O configuration (IO-Agent), PLC code generation (PLC-Agent), and, where applicable, retrieval of external knowledge (RAG-Agent) which the Orchestrator-Agent autonomously calls after planning the required process steps. This decomposition is motivated by the assumption that narrower agent roles can reduce prompt overload and improve task focus relative to monolithic architectures.

Architecture **A3** is the base multi-agent configuration consisting of an Orchestrator-Agent, an I/O-Agent and a PLC-Agent. By distributing responsibilities and MCP tools across agents, A3 is intended to mitigate the risk of prompt bloat and improve modular handling of tools calls and engineering subtasks. Architecture **A4** extends A3 by adding a RAG-Agent with access to a vector-based retrieval component. This enables semantic similarity search over vendor-specific knowledge like vendor manuals and over hardware-related project knowledge relevant to the use cases. The objective is to ground generation in document-based facts and thereby reduce hallucinations and vendor-specific mismatches [23].

A5 introduces the standardized A2A communication protocol to A3 to improve interoperability and reduce framework-specific coupling. Finally, **A6** combines both RAG and A2A.

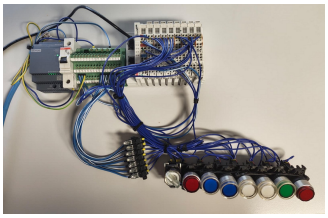
Together, A3–A6 enable a systematic comparison of knowledge grounding (RAG) and inter-agent interoperability (A2A), assessing whether these standardization layers justify their associated coordination and resource overhead in industrial settings.

IV. Experimental Setup

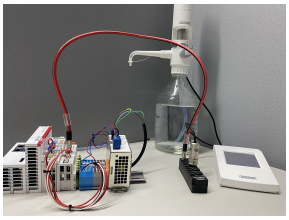
This section introduces two industrial use cases, the experimental setup and the evaluation criteria.

A. Industrial Use Cases

To evaluate the proposed framework (A0-A6), two distinct use cases (UC) with different characteristics and levels of complexity are employed. Based on interviews with automation experts from Boehringer Ingelheim conducted in a large-scale industrial engineering environment, each use case is estimated to require approximately four to ten days of total engineering effort, depending on the engineer's skill & experience level. This serves as human-baseline. UC1 addresses a leakage detection scenario, while UC2 focuses on a dispensing process which is based on the RS232 standard for serial communication. UC1 implements a signal-based control scheme. UC2 employs a state-based architecture that combines a control logic with several IEC 61131-3 function blocks. The hardware setting for UC1 is shown in Fig. 3a. A Beckhoff PLC equipped with an EtherCAT coupler EK1100, as well as EtherCAT's digital I/O modules EL1008 and EL2008 are used. The push-buttons are used to simulate events and triggers. The UC2 hardware setup, shown in Fig. 3b, comprises a Beckhoff IPC CX9240, Digital Input EL1809, an EtherCAT-P interface EK1322, an EtherCAT-P RS232 interface box, and a Hirschmann Opus dispenser. Both scenarios are representative of pharmaceutical-lab automation.



(a) Hardware Setting UC 1



(b) Hardware Setting UC 2

1) UC1 Leakage Box: The UC1 illustrates a leak-detection system consisting of six signal boxes. The system supports the overall safety concept of the plant but is not part of the functional safety system. For testing, the actual leakage sensors are replaced by push-buttons that emulate fault conditions, and the system monitors safety states such as off, on, diagnosis, ready/quit, and leakage shutdown. When a leak is detected, the control logic instantly isolates the voltage supply to guarantee safe operation. For this UC six base-function tests and three robustness tests are defined in Tab. II. The automated workflow starts with the generation of prompts that describe the required I/O configuration, followed by the creation of compilable IEC 61131-3 function blocks. These blocks are linked to global variables within the MAIN program and tested on the hardware setup.

2) UC2 RS232 interface: UC2 models a typical liquid-handling laboratory setup in which an RS-232-controlled dispenser is driven by three state-based FBs. The implementation focuses on a documented communication protocol, timeout- and error-handling via two-digit error codes. The first FB initializes the dispensing process. It configures the PC as Master, starts the ventilation routine, and reads the current process parameters. The second FB is responsible for the dispensing process. It receives the desired process parameters, waits for a start command, and then drives the dispenser. The third FB performs deinitialization by resetting the PC-Master flag, clearing all internal states, and releasing approval signals. For this use case, four nominal-operation tests and six robustness tests are defined in Tab. III.

Table II: Evaluation criteria for UC1

Nominal tests
1 Initialize plant: set state = active and clear all leakage alarms.
2 *Acknowledge* button → enable plant, turn on leakage detector, energize contactors.
3 Leak detection triggers → de-energize contactors (plant forced to safe state).
4 Leak clears without acknowledgement → keep fault flag set and contactors OFF.
5 Press *Acknowledge* after TL-04 → reset fault flag and re-energize contactors.
6 While a leak is active, *Acknowledge* must have no effect; plant stays safe.
Robustness tests
7 Two concurrent leaks remain active until both are cleared; after *Acknowledge* leakage detection resumes.
8 Leak → power-cycle plant → leak is fixed; contactors stay OFF until a *Quit* command is issued.
9 Leak appears during start-up; it must be cleared before *Acknowledge* can release detection.

Table III: Evaluation criteria for UC2

Nominal tests
1 Init via *Init* button → start-up telegrams are exchanged without failure
2 Set operational parameters with *Set* button → parameter telegrams are exchanged without failure
3 Start dispensing via *Dispense* → status telegrams continuously polled.
4 De-init with *De-init* → de-init telegrams are sent without failure.
Robustness test
5 Inject a timeout during dispensing; Timeout shown, dispensing stops; after *Acknowledge* the FB restarts.
6 Press *Init* again after a successful init; request ignored, no telegrams generated.
7 Attempt to start dispensing before parameters are set; the *Start* command is ignored; no telegrams are sent.
8 Issue *De-init* immediately after start; ignored (still initializing), no telegrams.
9 Press *Stop* during init; logic resets to base state, system can be started again without extra acknowledgment.

The agent must (1) split the protocol into three state-based FBs and generate their code, (2) map inputs to FB signals, (3) configure the RS-232 channel in MAIN, and (4) repeatedly compile until all syntactic and logical errors are eliminated.

B. Benchmark Technologies

The experiments were run with the Google ADK [24] because it logs system prompts, user messages, assistant replies, timestamps and token usage for every call. Combined with the custom proxy layer, which captures request and response data

Table IV: Evaluation of AC1/AC2 for UC1 & UC2 (Claude = C, GPT = G).
✓ = AC1/AC2 fulfilled; × = not fulfilled; ○ = syntactically correct but semantic test failed/partially succeeded.

Run	A0		A1		A2		A3		A4		A5		A6	
	C	G	C	G	C	G	C	G	C	G	C	G	C	G
Use Case 1														
1	○	○	✓	○	✓	✓	✓	✓	-	-	✓	✓	-	-
2	○	○	✓	○	✓	✓	✓	✓	-	-	✓	✓	-	-
3	○	○	✓	○	✓	✓	✓	✓	-	-	✓	✓	-	-
4	○	○	✓	○	✓	✓	✓	✓	-	-	✓	✓	-	-
5	○	○	✓	○	✓	✓	✓	✓	-	-	✓	✓	-	-
Use Case 2														
1	×	×	○	×	○	×	○	×	○	○	-	-	○	○
2	×	×	○	×	○	×	○	×	✓	○	-	-	○	○
3	×	×	○	×	○	×	✓	×	○	○	-	-	○	○
4	×	×	○	○	○	×	○	×	○	○	-	-	○	○
5	×	×	○	×	○	×	○	×	○	○	-	-	○	×

at the API boundary, this setup provides end-to-end observability across the entire agent pipeline. This level of observability is important for benchmarking because it supports traceability and enables the analysis of non-functional metrics such as token consumption and interaction overhead. The evaluated foundation models were Claude Opus 4.6 and GPT-5.1 because Claude Opus 4.6 is specialized for coding skills [25] while ChatGPT 5.1 is designed for general purpose dialogues with a user [26]. This dual-model approach was adopted to determine whether the performance of the proposed framework is sensitive to the specific characteristics of the underlying LLM. The development environment uses a beta version of Beckhoff's TwinCAT-CoAgent [27], which partially exposes the TwinCAT API via custom MCP servers, supplying the tools needed for PLC code generation and automation-project configuration. For the RAG-enabled architectures, the retrieval component indexes TwinCAT documentation regarding the TF6340 serial communication with explanation of the FBs. Evaluation combines functional and non-functional metrics: Functional performance is measured by Acceptance Criterion 1 (AC1), syntactic correctness (successful TwinCAT compilation) and Acceptance Criterion 2 (AC2), semantic correctness (pre-defined use-case tests in Tables II and III). Non-functional performance includes tool calls, token usage, and total cost across repeated runs.

Each architecture-model-use-case condition was executed multiple times with a fixed task prompt and identical evaluation criteria. The workflow consists of prompting, automated generation, logging, and manual inspection of artifacts and hardware test results. When syntactic failures occurred, manual remediation was applied only as a secondary diagnostic step to isolate syntactic from semantic limitations. Task prompts (one per use case) embed the hardware setting and required engineering artifacts described earlier. Full prompts, run logs, model parameters and measured values are available at https://github.com/KIT-IRS/PLCAgentBenchmark.

V. Evaluation

A. Results

The results for UC1 and UC2 are summarized in Tab. IV.

Although comparing foundation models is not the primary objective of this benchmark, the results show that model choice is a first-order determinant of performance. Across both use cases, Claude shows a better performance with respect to AC1 and AC2. In UC1, this difference is most visible in the baseline and tool-enabled settings A0 and A1, where Claude achieves syntactic correctness more often, whereas GPT repeatedly fails because of hardware-binding and notation errors. From A2 onward, the gap narrows in UC1 because both models satisfy the reported acceptance criteria. In UC2, however, the model effect becomes decisive: GPT fails to compile syntactically correct code without a RAG extended architecture, whereas Claude achieves AC1 in single-agent and multi-agent architectures and reaches AC1 consistently from A2 onward. Thus, architectural refinements cannot fully offset an inappropriate LLM, and future improvements in foundation models may lessen the need for increasingly complex agentic architectures. That said, agentic orchestration architectures retain clear relevance: they provide the determinism and control guardrails that probabilistic models cannot guarantee on their own. Furthermore, complex automation engineering projects are fundamentally unsuited to a single-call paradigm. They demand structured orchestration with traceable, auditable task chains. The error patterns also indicate that performance depends significantly on the initial task prompt. For Use-Case 2, the same error patterns were observed for both ChatGPT and Claude: incorrect string indexing due to missing platform-specific knowledge and flawed receive logic caused by insufficient understanding of cyclic PLC execution. This suggests that the prompt must be formulated more precisely to ensure the generation of semantically correct code highlighting the importance of standardized information models between lifecycle steps that are testable. The benchmark design enables five main comparisons:

1) A0 versus A1 isolates the effect of tool access,
2) A1 versus A2 isolates the effect of an explicit closed-loop agent,
3) A2 versus A3 captures the transition from a single-agent to a multi-agent structure,
4) A3 versus A4 probes the incremental effect of RAG,
5) and A3/A4 versus A5/A6 probes the effect of standardized inter-agent communication.

UC1 shows that simple agents perform adequately, making RAG overhead irrelevant. In UC2, external knowledge is required, so RAG-enhanced architectures are evaluated, excluding non-RAG configuration A5. An improvement from A3 to A4 confirms the necessity of RAG.

For Claude, the comparison in UC1 between A0 and A1 shows that tool access improves syntactic correctness in both use cases. However, A1 alone does not ensure semantic correctness, which indicates that tool access without stronger control logic remains insufficient. The clearest improvement appears from A1 to A2: a persistent single-agent loop makes both AC1 and AC2 consistently positive in UC1, proving that a closed-loop agent adds value beyond mere tool integration.

These varying results for A3 and A4 in UC2 with Claude underscore the non-deterministic nature of LLMs and agentic AI approaches and indicate that multi-agent structures are not inherently superior. As the underlying LLM quality improves, lightweight agent designs may offer performance comparable to more complex multi-agent systems, and in some cases to single-agent structures. To further examine this point, efficiency metrics are evaluated for the Claude-based approach.

Figures 4–6 display the resource efficiency metrics and introduce a comparison of results as box plot. The token amount is composed of the input tokens, cached write and read tokens, reasoning tokens and output tokens. The cost is all summed from the total LLM and RAG costs. Although a sample of only five tests lacks sufficient statistical power to reach significance, the data nevertheless indicate a clear trend.

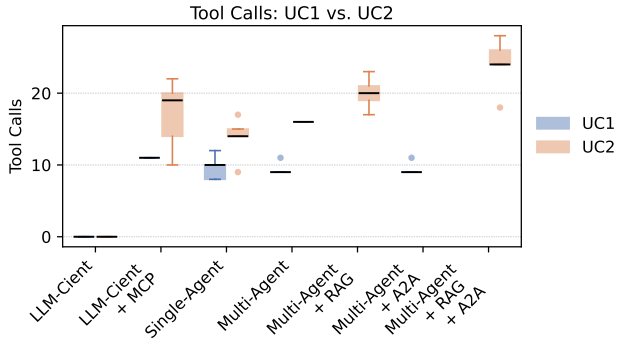


Figure 4: Comparison UC1 and UC2: Tool Calls

Figure 4 depicts the tool calls per architecture. For UC1, the single-agent and multi-agent structures show a broadly similar low level of tool use around 10. For UC2, tool use increases across most architectures, especially in the RAG-enabled multi-agent architectures with a median of 20. The MCP-enabled LLM in UC2 shows comparatively high variability, which is consistent with the non-deterministic behavior already visible in the functional results.

Regarding the total token consumption depicted in Figure 5, it can be observed that usage peaks in the single-agent architecture. However, with the transition to multi-agent architectures, the token count decreases and remains approximately constant. For UC2, token usage generally increases with architectural complexity. This trend is less pronounced in UC1, whose lower task complexity requires less context and fewer iterative interactions than UC2.

Overall cost also increases with architectural complexity. A plausible explanation is that reasoning and cached-token usage increase with inter-agent negotiation and retrieval overhead, which is reflected in Fig. 6.

Taken together with AC1 and AC2, these non-functional results indicate that A2 currently offers the strongest correctness-resource trade-off in the reported benchmark. Relative to A0 and A1, it substantially improves functional performance,

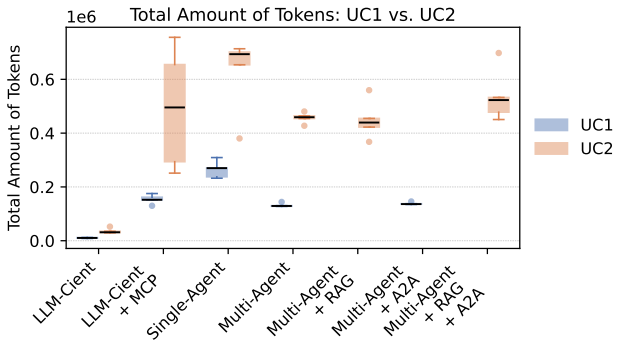


Figure 5: Comparison UC1 and UC2: Total Amount of Tokens

while relative to A3-A6 it avoids part of the additional coordination overhead visible in tool calls, token usage, and cost. Stronger claims about the incremental benefit of RAG or A2A require more complex use cases. Multi-Agent Architectures, implemented with A2A and RAG, might be beneficial in case of usage of heterogenous vendor and vendor-specific agents in order to standardise the communication (A2A) or retrieve suitable knowledge (RAG). This example shows an homogenous environment.

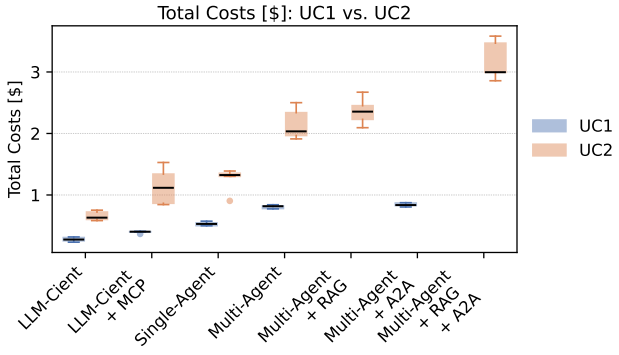


Figure 6: Comparison UC1 and UC2: Total Costs

Since these engineering tasks correspond to several days of manual work, the reported results already indicate substantial efficiency gains, especially for the single-agent configuration. At the same time, a rigorous human-versus-AI productivity and cost comparison remains outside the scope of the present study and should therefore be addressed in future work.

B. Discussion

The results show that increased architectural complexity does not automatically improve PLC engineering performance. The strongest supported comparison in the current benchmark is A1 versus A2: across both use cases, a closed-loop single agent transforms tool access from a fragile capability into a more reliable engineering process. This directly supports the central claim that tool access alone is insufficient without a control strategy that can iteratively interpret failures and refine outputs. The benefit of RAG can be seen in combination with

GPT since it enables syntactically correct code generation. By contrast, A2A effects remain preliminary because the current dataset is not complex enough. Multi-agent variants remain relevant because they improve modularity, separation of concerns, and interoperability through standardized interfaces. However, in the present benchmark, RAG and A2A increase tool use, token consumption, and cost without uniformly improving correctness.

These conclusions are limited to two use-cases in a single vendor environment and a beta toolchain, and they are performed as single shot experiments rather than a human-in-the-loop approach. Future work should broaden the benchmark to more vendors and task families to pinpoint when multi-agent decomposition becomes advantageous.

VI. Conclusion

This work introduces and evaluates seven agentic AI architectures for two industry use cases. Within the quantitative scope of AC1, AC2, tool calls, token usage, and cost, the results suggest that a closed-loop agent approach is more important than architectural complexity alone: This study demonstrates that the most significant performance gains are achieved through vendor-specific tool access and explicit closed-loop orchestration within a single-agent architecture. Conversely, multi-agent and RAG-enhanced variants failed to provide consistent additional benefits, suggesting that the increased coordination overhead outweighs the performance gains at the current level of task complexity. The experimental results emphasize that the choice between a single-agent and a multi-agent structure depends on the underlying application. In both examined use-cases, the single-agent shows the best trade-off between resource efficiency, deterministic result and correctness. Nevertheless, this also needs to be part of future research in which more complex use-cases are examined in which a multi-agent system is assumed to outperform a single-agent due to the context complexity. Future research will investigate how a targeted reduction and improvement of project-specific prompt components influences the quality of generated artifacts, with the aim of quantifying the impact of prompt trimming on engineering outcomes. By systematically varying prompt size and content, the trade-off between prompt complexity, maintainability, and the level of knowledge provisioning embedded in the model will be analyzed, providing insights into potential efficiency gains across different engineering skill levels. Further research on agentic AI across lifecycle phases (e.g., design and implementation) is needed, including evaluating how industrial standards can be integrated as testable artifacts. The explicit quantification of human engineering time is left for future work. Future studies should extend this methodology to brownfield contexts where agentic AI must negotiate legacy constraints while preserving operational continuity.

References

[1] G. Gutermuth, B. Schroeter, R. Drath, N. Li, and C. Messinger, "A novel approach to measure engineering efficiency of automation projects," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pp. 1–8.

[2] L. Dürkop, L. Wisniewski, S. Heymann, B. Lücke, and J. Jasperneite, "Analyzing the engineering effort for the commissioning of industrial automation systems," in *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 1–4.

[3] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The Impact of AI on Developer Productivity: Evidence from GitHub Copilot."

[4] 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 A Multi-Agent Collaborative Framework.

[5] M. Gao, Y. Li, B. Liu, Y. Yu, P. Wang, C.-Y. Lin, and F. Lai. Single-agent or Multi-agent Systems? Why Not Both?

[6] M. Fakih, R. Dharmaji, Y. Moghaddas, G. Q. Araya, O. Ogundare, and M. A. A. Faruque, "LLM4PLC: Harnessing Large Language Models for Verifiable Programming of PLCs in Industrial Control Systems," 2024.

[7] 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," 2024.

[8] M. Klesel and H. F. Wittmann, "Retrieval-Augmented Generation (RAG)," vol. 67, no. 4, pp. 551–561.

[9] D. Yang, A. Wu, T. Zhang, L. Zhang, F. Liu, X. Lian, Y. Ren, J. Tian, and X. Che. AutoPLC: Generating Vendor-Aware Structured Text for Programmable Logic Controllers.

[10] J. Kersting, M. Rummel, and G. Benndorf, "Vendor-Aware Industrial Agents: RAG-Enhanced LLMs for Secure On-Premise PLC Code Generation," Jan. 2026.

[11] Anthropic. (2024) Model context protocol. https://github.com/modelcontextprotocol. Accessed 13.02.2026.

[12] Overview - A2A Protocol. Accessed: Apr. 10, 2026. [Online]. Available: https://a2a-protocol.org/latest/specification/

[13] H. Koziolek, S. Gruener, and V. Ashiwal, "ChatGPT for PLC/DCS Control Logic Generation," May 2023.

[14] H. Koziolek and A. Koziolek. LLM-based Control Code Generation using Image Recognition.

[15] H. Koziolek, S. Grüner, R. Hark, V. Ashiwal, S. Linsbauer, and N. Eskandani, "LLM-based and Retrieval-Augmented Control Code Generation," in *Proceedings of the 1st International Workshop on Large Language Models for Code*. ACM, pp. 22–29.

[16] Y. Zhang and M. Sousa, "Exploring LLM Support for Generating IEC 61131-3 Graphic Language Programs."

[17] L. Ren, H. Wang, J. Dong, H. Wang, S. Liu, Y. Laili, and L. Zhang, "MetaIndux-PLC: A Control Logic-Guided LLM for PLC code generation in industrial control systems," *Applied Soft Computing*, vol. 184, p. 113673, Dec. 2025.

[18] H. Koziolek, T. Braun, V. Ashiwal, S. Linsbauer, M. A. Hansen, and K. Grotterud, "Spec2Control: Automating PLC/DCS Control-Logic Engineering from Natural Language Requirements with LLMs - A Multi-Plant Evaluation," 2025.

[19] Y. Zhang, W. Xia, B. Zhao, T. Yuan, and X. Yu, "MPC-Coder: A Dual-Knowledge Enhanced Multi-Agent System with Closed-Loop Verification for PLC Code Generation," vol. 18, no. 2, p. 248.

[20] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian. A Comprehensive Overview of Large Language Models.

[21] D. B. Acharya, K. Kuppan, and B. Divya, "Agentic AI: Autonomous Intelligence for Complex Goals—A Comprehensive Survey," vol. 13, pp. 18 912–18 936.

[22] T. Gan and Q. Sun. RAG-MCP: Mitigating Prompt Bloat in LLM Tool Selection via Retrieval-Augmented Generation.

[23] 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.

[24] Multi-agent-systeme mit dem adk erstellen. Google Codelabs. Accessed: 2026-04-11. [Online]. Available: https://codelabs.developers.google.com/codelabs/production-ready-ai-with-gc/3-developing-agents/build-a-multi-agent-system-with-adk?hl=de

[25] Claude Opus 4.6. Accessed: 2026-04-22. [Online]. Available: https://www.anthropic.com/news/claude-opus-4-6

[26] Gpt-5.1: Ein intelligenteres, stärker dialogorientiertes chatgpt. OpenAI. [Online]. Available: https://openai.com/de-DE/index/gpt-5-1/

[27] B. Automation. (2024) Mit erweiterten ki-funktionen noch einfacher optimierungspotenzial erschließen. Beckhoff Automation GmbH & Co. KG. Accessed: 2026-04-15. [Online]. Available: https://t1p.de/bewmz