PLCWORLD: BENCHMARKING LLM-GENERATED PLC PROGRAMS IN CLOSED-LOOP PLANT SIMULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Programmable logic controllers (PLCs) coordinate industrial equipment by reading sensor inputs and issuing control commands. Evaluating whether large language model (LLM)-generated PLC programs satisfy task requirements and safety constraints requires observing how their commands affect device and workpiece states. We introduce **PLCWorld**, a common closed-loop execution environment and benchmark that couples Structured Text (ST) execution with simulated plant responses and sensor feedback. Grounded in control relations identified in industrial PLC programs and engineering documentation, **PLCWorld** contains 100 synthetic tasks and 473 registered task–condition pairs across *Motion Control* and *Material Handling*, with difficulty defined by control-dependency scope. A common protocol reports *Task Success* and *Safety Violation* separately. Validation combines practitioner review, reference and alternative programs, targeted counterexamples, specification–evaluator alignment checks, and comparisons with independent ST runtimes. Reference and alternative programs satisfy their applicable cases, while all 542 targeted counterexamples activate their designated evaluator rules under at least one registered condition. *Execution Gap* relates submission-profile acceptance to subsequent task failure or observed *Safety Violation*. Across the constructed task groups, direct GPT-5.5 achieves 82.70% *Task Success* on Easy cases but 25.10% on Hard cases. Evaluations of six LLMs and four adapted generation-and-verification workflows further expose differences between completion, safety, and generation cost.

1 INTRODUCTION

A Programmable Logic Controller (PLC) is an industrial controller that coordinates a production process by reading sensor signals and controlling devices such as motors, conveyors, and grippers (Vyatkin, 2013). PLC control involves dependencies within a basic control function, between functions of a job, and between jobs that share equipment. These dependencies arise from device response timing, required device and workpiece states, and access to shared resources. They determine when an operation may start, which states must be maintained during execution, and when the operation can be considered complete. Failing to respect these dependencies can prevent task completion or violate safety constraints. Such concerns have long motivated research on the formal verification of PLC programs (Ovatman et al., 2016; Adiego et al., 2015).

To express such control logic, engineers write programs in the IEC 61131-3 languages (International Electrotechnical Commission, 2013), among which Structured Text (ST) is a high-level textual language. Recent work generates ST code from natural-language requirements with large language models (LLMs), using document and code-example retrieval (Koziolek et al., 2026; Yang et al., 2025), compiler feedback (Fakih et al., 2024), and formal verification (Liu et al., 2026) to support generation and revision. Evaluation has also expanded to project context and execution outcomes (Tu et al., 2026). The evidence provided by these evaluations depends on the properties, inputs, and execution conditions they cover. We focus on evaluating whether an LLM-generated ST program can coordinate the required operations to complete a plant task while satisfying its safety constraints under specified operating conditions.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

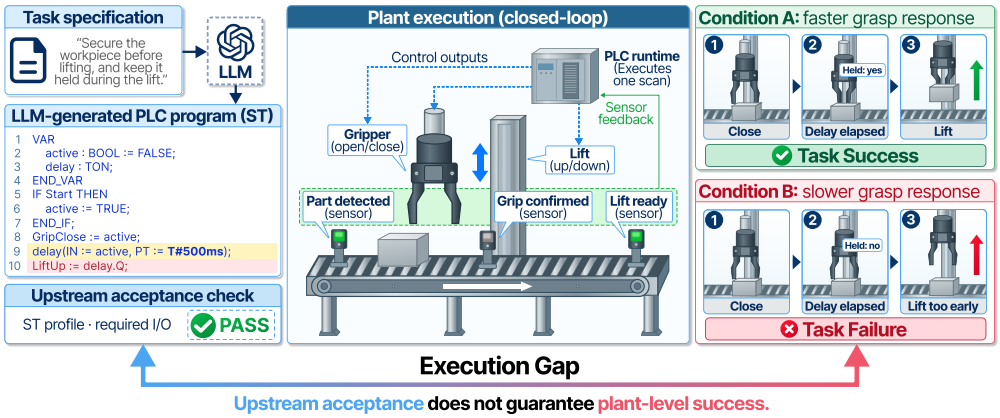

Figure 1: ***Execution Gap.*** Even after the specified upstream checks pass, the lift can rise without the workpiece if grasping is incomplete when the fixed wait ends.

On physical equipment, issuing a command does not imply that the corresponding operation has completed. Whether a dependent operation can start or continue depends on the actual state of the controlled equipment and process, collectively called the plant. The plant state evolves as devices respond to commands and interact with workpieces. A PLC program must therefore coordinate its commands with the device and workpiece states required by the task.

Figure 1 illustrates this dependency with a gripper that must secure a workpiece before lifting it. The generated ST program commands the gripper to close, waits for a fixed interval, and then commands the lift. If grasping takes longer than this interval, even within the specified operating range, the lift starts before the workpiece is secured, leaving it on the table. The failure arises because the program advances to lifting without satisfying the prerequisite grasp state, despite issuing the commands in the intended order.

In plant operation, the outcome of one operation can determine the conditions under which subsequent operations may proceed. Evaluating these dependencies calls for closed-loop plant execution, in which the program's control outputs change device and workpiece states, and sensor inputs derived from those states are returned to the program for subsequent control. An evaluation environment must sustain this interaction throughout the task so that task completion and safety can be assessed from the resulting plant behavior.

We introduce **PLCWorld**, an execution environment and benchmark for evaluating LLM-generated ST programs on industrial control tasks. Our contributions are as follows. (1) We develop a closed-loop plant execution environment that couples ST execution with simulated device and workpiece behavior at each PLC scan. (2) We construct *Motion Control* and *Material Handling* task suites grounded in control relations identified in industrial PLC programs and engineering materials, with explicit task objectives, control dependencies, operating conditions, and safety constraints. (3) We establish a common evaluation protocol that reports *Task Success* and *Safety Violation* separately and compares upstream checks with plant execution outcomes on the same submissions to analyze *Execution Gap*.

We evaluate six LLMs and four generation-and-verification methods on PLCWorld. Across both settings, Hard tasks show substantially lower *Task Success* rates than Easy and Medium tasks, while *Safety Violation* captures failures not reflected by completion alone. The evaluated adaptations show mixed completion and safety changes versus matched direct generation, without consistent plant-level gains. Substantial failures persist after common upstream acceptance. Failure analysis identifies failure to establish valid completion by the deadline as the most frequent recorded signature.

2 Related Work

LLM-based PLC code generation. Recent work has explored LLM-based generation and validation of PLC control code using retrieval, compiler feedback, formal verification, expert re-

Table 1: Reported validation coverage of LLM-based PLC code generation studies.

Study	Program validation		Dependency validation		Plant outcomes	
	Syntax / build	Program execution	Control dependencies	Workpiece handling	Task completion	Safety violations
LLM4PLC	○	○	○	△	△	×
Agents4PLC	○	○	○	×	×	×
AutoPLC	○	○	○	×	×	×
Spec2Control	△	×	△	×	×	×
SemaPLC	○	○	○	×	×	×
PLCWorld	○	○	○	○	○	○

○: assessed; △: partially assessed; ×: not reported.

view, and runtime testing. Representative approaches include LLM4PLC (Fakih et al., 2024), Agents4PLC (Liu et al., 2026), AutoPLC (Yang et al., 2025), Spec2Control (Koziolek et al., 2026), and SemaPLC (Tu et al., 2026). Table 1 summarizes the validation scope reported by these studies.

Prior studies assess control dependencies through formal verification and runtime tests, with LLM4PLC also reporting a physical handling demonstration. The plant-outcome columns concern task completion and safety judgments derived from physical or simulated plant states and events, beyond agreement between program outputs and reference traces. PLCWorld brings device and workpiece dependencies into a common closed-loop execution environment and evaluates *Task Success* and *Safety Violation* separately under specified operating conditions.

PLC development and commissioning tools. Existing PLC tools support controller simulation (Mitsubishi Electric, 2023), virtual commissioning with equipment models (Mitsubishi Electric, n.d.; Siemens, 2024), and execution of IEC 61131-3 programs through open runtimes (Alves & Morris, 2018; Autonomy Logic, n.d.). Comparing LLM-generated programs using these tools requires common tasks, operating conditions, and criteria for task completion and safety. PLCWorld combines a closed-loop plant execution environment with task suites organized by control dependencies and a common evaluation protocol, enabling systematic comparisons across models and generation-and-verification methods.

Execution-based evaluation. Recent work evaluates LLM outputs through their consequences in executable environments. BIRD-INTERACT (Huo et al., 2026) evaluates interactive database tasks, BuildArena (Xia et al., 2025) tests generated engineering structures in physics simulation, and MC-Coder (Li et al., 2025) verifies generated Python motion-control code in simulation. For PLC programs, such evaluation must preserve the control dependencies through which outputs change device and workpiece states and those states, in turn, constrain subsequent control.

3 PLCWORLD

PLCWorld combines a plant execution environment, benchmark task suites, and an evaluation protocol for LLM-generated ST programs (Figure 2). Task specifications define the required outcomes and operating conditions, and submitted programs are evaluated through interaction with a simulated plant. The resulting execution records are used to assess *Task Success* and *Safety Violation* separately.

3.1 PLANT EXECUTION ENVIRONMENT

PLC operations depend on device and workpiece states that determine when they may start, continue, or complete. Issuing a command does not necessarily establish the required state, so evaluating these dependencies requires following the command's effects through plant execution. PLCWorld couples each generated ST program to a simulated plant that updates device and workpiece states and returns sensor inputs derived from those states for subsequent control.

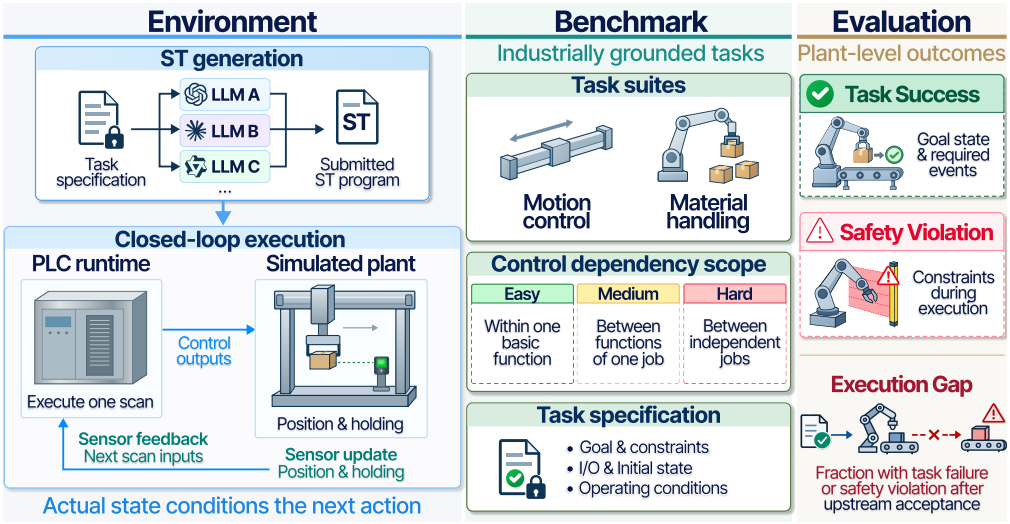

Figure 2: **The PLCWorld framework.** ST programs generated from task specifications are executed in closed loop with a simulated plant. Execution records are used to assess *Task Success* and *Safety Violation* separately, and comparison with upstream checks on the same programs reveals *Execution Gap*.

Closed-loop execution. Each execution starts from an ST program, a device configuration with I/O bindings, an initial plant state, and execution conditions. The ST program runs cyclically in PLC scans, reading sensor inputs and computing control outputs while retaining its internal state across scans. After each scan, the plant advances under these outputs, and sensor observations of its updated state provide inputs for the next scan. Equation (1) expresses this cycle over a scan interval Δt, where X_t, Y_t, C_t, and P_t denote sensor inputs, control outputs, retained program state, and plant state; θ_t specifies external execution conditions, F updates the plant state, and H derives sensor observations.

$$(C_{t+1}, Y_t) = \mathrm{Exec}(C_t, X_t; \Delta t), \quad P_{t+1} = F(P_t, Y_t, \theta_t, \Delta t), \quad X_{t+1} = H(P_{t+1}, \theta_{t+1}). \tag{1}$$

Timers advance by the scan interval Δt. External inputs, such as start signals and job requests, are specified by the execution conditions, whereas sensor feedback is derived from the updated plant state rather than replayed from a prerecorded trace. The supported ST profile specifies the supported ST syntax, function-block rules, and initialization.

Plant state. The plant state P_t represents device positions and operating states, workpiece locations, and holding and support relations. An acquisition or handover takes effect only when the device state and workpiece location satisfy the required conditions. For example, if a workpiece is outside the gripper's acquisition region, closing the gripper establishes no holding relation, and subsequent motion leaves the workpiece behind. The simulator models device operation and response times, sensor feedback, and the transitions for acquisition, holding, and handover, but not contact forces or mechanical deformation.

Execution records. Execution runs to the evaluation endpoint or termination condition regardless of completion claims. At each scan, PLCWorld records sensor inputs X_t, control outputs Y_t, the updated plant state P_{t+1}, and events. Program errors are recorded with their interruption points. These records support the separate assessment of *Task Success* and *Safety Violation*, failure analysis, and 3D replay.

3.2 Benchmark Tasks

Task specification. Each task specifies a goal plant state, required intermediate events, and their precedence relations. Safety constraints are stated separately. Any implementation satisfying these requirements is valid, regardless of its internal states, branching, or timers.

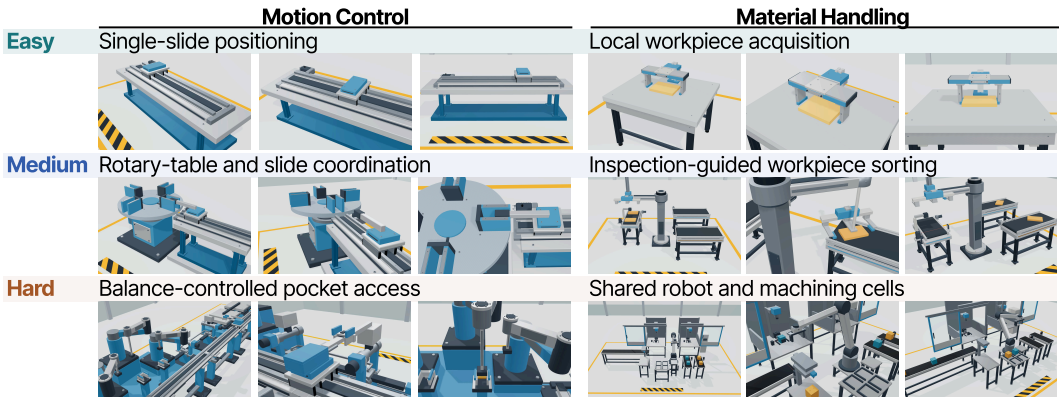

Figure 3: **Representative executions by suite and difficulty.** Columns show *Motion Control* (left) and *Material Handling* (right), and rows show control dependencies within one basic function (Easy), across the functions of one job (Medium), and between jobs sharing resources (Hard).

Table 2: Difficulty levels defined by dependency scope, with tasks and representative examples.

Level	**Tasks**[a]	**Dependency Scope**	***Motion Control***	***Material Handling***
Easy	15	Within one basic function	"*Move a slide to a position, confirm arrival and standstill*"	"*Grasp a workpiece at a prepared position, confirm it is held*"
Medium	20	Between functions of one job: one function's state conditions another's execution	"*Retract the slide, reorient the table, approach after standstill*"	"*Transport a workpiece, release it after the jig is ready*"
Hard	15	Between independent jobs: one job's state constrains another's progress	"*Coordinate two lift jobs as the shared shuttle and rotary/approach cell become available*"	"*Coordinate two handling jobs as the shared robot and inspection station become available*"

[a] Per suite. PLCWorld contains 100 tasks in total.

The model receives these requirements together with device relationships, I/O semantics, the initial plant state, and permitted operating conditions. The specification explains device responses to output activation and deactivation and sensor indications of position, operation completion, and workpiece holding. Evaluation conditions may vary the initial state, device responses, or external inputs within the specified operating range. The specification distinguishes the completion deadline from the evaluation endpoint or termination condition and states any requirement to maintain the completed state.

Task suites. Drawing on established work in *Motion Control* (Ruderman et al., 2020) and PLC-based *Material Handling* (Bassily et al., 2007), we organize tasks into two suites by their primary functional outcome. (1) *Motion Control* tasks control device positions and operating states, such as positioning, homing, and coordinated movement. (2) *Material Handling* tasks change workpiece locations and holding relations, such as acquisition, transport, placement, and handover. For example, a transport device that arrives without its workpiece fails the objective.

Difficulty levels. We define difficulty by the scope of control dependencies that a program must coordinate, using the same criterion for both task suites. These levels distinguish coordination within a basic function, between functions, and between jobs, capturing differences in control requirements that device count or code length alone does not describe. The basic unit is a control operation with its own objective and completion condition. A basic function includes its sensor checks, short sequences, and retained state, with boundaries fixed during task construction. Table 2 defines the levels and provides task counts and examples, while Figure 3 illustrates representative executions. Short sequences or state memory confined to one basic function remain Easy, and serial or fixed-priority control remains valid for Hard tasks if it meets the requirements.

3.3 EVALUATION

PLCWorld evaluates functional completion (*Task Success*) separately from observed breaches of safety constraints (*Safety Violation*). A case is an assigned evaluation of a task's submitted program under one registered execution condition. For each independent repetition, one submitted ST program per task is held fixed and evaluated across all its registered conditions, yielding 473 cases for 100 tasks. A case may succeed despite a violation or fail without one.

Task Success. We assign $U_e = 1$ when the plant reaches the required goal state within the deadline while satisfying all required events, precedence relations, and hold conditions. When a completion signal is required, the plant state after applying the signal for one scan must satisfy the completion conditions. *Success Rate* (SR) in Eq. (2) averages $U_e \in \{0, 1\}$ over $\mathcal{E}$, the set of evaluation cases excluding verified infrastructure errors.

Safety Violation. An execution incurs a violation if it breaches a specified constraint during evaluation, such as conflicting outputs, operation without an interlock, or prohibited movement. Monitoring continues past completion claims, and recorded violations persist. *Safety Violation Rate* (SVR) in Eq. (2) averages the indicator $V_e \in \{0, 1\}$ over $\mathcal{E}_{\text{exec}} \subseteq \mathcal{E}$, cases entering plant execution. Violations before interruption count, and observing none does not establish safety over the full window.

$$\text{SR} = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} U_e, \qquad \text{SVR} = \frac{1}{|\mathcal{E}_{\text{exec}}|} \sum_{e \in \mathcal{E}_{\text{exec}}} V_e. \tag{2}$$

Missing or unusable outputs, submission-profile violations, parsing failures, and execution errors attributable to the model count as Task Success failures. Non-executed cases are excluded from SVR, while infrastructure errors are excluded from all metrics and reported separately.

Execution Gap. For an upstream check k, ExecutionGap_k in Eq. (3) is the fraction of included cases $\mathcal{A}_k \subseteq \mathcal{E}$ assigned to programs passing k that show task failure $\mathcal{F} = \{e \in \mathcal{E} : U_e = 0\}$ or an observed violation $\mathcal{W} = \{e \in \mathcal{E}_{\text{exec}} : V_e = 1\}$. Cases with both count once, each check is computed separately, and each check and the plant evaluation use the same submitted program.

$$\text{ExecutionGap}_k = \frac{|\mathcal{A}_k \cap (\mathcal{F} \cup \mathcal{W})|}{|\mathcal{A}_k|}. \tag{3}$$

4 BENCHMARK CONSTRUCTION AND VALIDATION

4.1 INDUSTRIALLY GROUNDED TASK CONSTRUCTION

PLCWorld grounds its task construction in PLC programs and engineering documentation from real-world manufacturing equipment. The source archive contains 2,201 PLC and related automation project files, together with I/O lists, electrical drawings, equipment manuals, and other engineering documents. We examine these programs and documents to identify recurring control relations, including feedback-conditioned actuation, access and clearance interlocks, support and handover relations, dependencies involving workpiece states, and coordination through shared equipment.

Each control relation records an action, its observable state or feedback, the device or workpiece state required before the action may proceed, and any state that must remain valid during subsequent operations. We instantiate these relations as synthetic tasks in a common execution environment, defining task goals, operating conditions, device arrangements, I/O mappings, and numerical parameters without reproducing the source implementation. Figure 4 summarizes the requirement families represented in these tasks. These families characterize task content rather than difficulty, which is defined by dependency scope. Their suite–difficulty breakdown is reported in Appendix B.1.

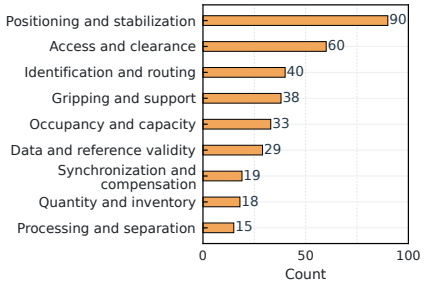

Figure 4: Control requirement families

4.2 BENCHMARK VALIDATION

Reliable evaluation requires plausible, sufficiently specified tasks, evaluator judgments consistent with requirements, and reproducible execution with consistent ST semantics. Task feasibility alone does not establish that the evaluator accepts alternative valid implementations or detects violations. As summarized in Table 3, we examine these requirements through four complementary validation checks, each addressing a distinct aspect of benchmark reliability. Detailed procedures and results appear in Appendix F.

Table 3: Overview of validation

Validation target	Validation method
(1) Task specification	Practitioner review
(2) Evaluator judgment	Reference/alternative ST & counterexamples
(3) Spec–evaluator alignment	Binding & semantic checks
(4) ST runtime	Reruns & cross-runtime checks

(1) Task specification validation. Practitioner review assesses whether the synthetic tasks are industrially plausible and sufficiently specified for implementation. Four practitioners experienced in PLC control and equipment operation review all 100 task specifications, providing assessments for the six suite–difficulty groups and recording individual task issues. They examine information sufficiency, completion and safety clarity, and dependency-based difficulty levels. Their feedback informs revisions before specifications are frozen for evaluation.

(2) Evaluator judgment validation. Reference ST programs written to satisfy the task specifications achieve *Task Success* without an observed *Safety Violation* across all 100 tasks and 473 registered task–condition cases. Alternative implementations for a subset of tasks yield the same outcomes in their tested executions, supporting acceptance beyond a single solution. Successful executions alone do not establish violation detection, so we also test controlled counterexamples targeting completion, precedence, retention, interlock, command, and safety requirements. Each triggers its prespecified evaluator rule in at least one registered execution condition.

(3) Specification–evaluator alignment validation. The evaluator should enforce published task requirements without introducing unstated conditions. A bidirectional audit checks whether requirements are represented in executable rules and whether those rules are supported by the specifications. Automated checks verify task-input integrity and consistency of I/O interfaces, scan periods, evaluator bindings, and submission constraints. Audit findings guide revisions and regression checks.

(4) ST runtime validation. Reference-case reruns reproduce trace hashes and judgments. Diagnostics use MATIEC/GCC (beremiz, 2017); S7-PLCSIM Advanced (Siemens AG, 2024) tests examine on-delay timer (`TON`) scan-boundary behavior. MATIEC/GCC comparisons cover 268 unchanged programs from six direct-generation models across all registered conditions with the frozen plant and evaluator. Traces agree in $1{,}162/1{,}199$ cases, *Task Success* in $1{,}195$, and *Safety Violation* in all. Four *Task Success* differences involve one program calling a `TON` instance twice per scan.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

We evaluate two settings. **(1) Direct ST generation** compares GPT-5.5 (OpenAI, 2026), GPT-4o (Hurst et al., 2024), Claude Sonnet 5 (Anthropic, 2026), Gemini 3.1 Pro (Google DeepMind, 2026a), Gemini 3.8 Flash (Google DeepMind, 2026b), and Qwen3.5-Plus (Qwen Team, 2026). Each produces three independent single-turn generations per task without tools, execution feedback, revision, or best-of selection. **(2) Generation-and-verification** compares PLCWorld adaptations of LLM4PLC (Fakih et al., 2024), Agents4PLC (Liu et al., 2026), AutoPLC (Yang et al., 2025), and SemaPLC (Tu et al., 2026) with GPT-5.5 as the shared backbone (Appendix G.3).

Each final program is evaluated under all registered task conditions at the task-specific PLC scan period. The common upstream check assesses final submissions against the supported ST profile and required I/O interface, separately from framework-internal compilation and verification. SR, SVR, and ExecutionGap follow Eqs. (2) and (3). We measure generation cost on a fixed, shared 18-task subset balanced across the six suite–difficulty groups. Details appear in Appendix G.

Table 4: Results (%) by suite and difficulty. *Execution Gap* is pooled across suites, conditional on common upstream acceptance. Bold marks each block's highest SR and lowest SVR, including ties.

Model / Method	Motion Control						Material Handling						Execution Gap↓
	Easy		Medium		Hard		Easy		Medium		Hard		
	SR↑	SVR↓	SR↑	SVR↓	SR↑	SVR↓	SR↑	SVR↓	SR↑	SVR↓	SR↑	SVR↓	
(1) Direct ST generation													
GPT-5.5	71.11	1.67	75.58	**7.05**	**18.06**	65.19	97.83	**0.00**	93.06	1.77	27.87	**32.24**	37.88
GPT-4o	14.44	22.37	8.91	42.06	0.00	**28.57**	28.99	13.49	1.04	21.11	0.00	57.71	92.75
Claude Sonnet 5	75.00	2.27	75.91	25.50	0.00	73.33	88.41	0.72	77.43	1.04	4.37	51.72	50.90
Gemini 3.1 Pro	78.33	**0.00**	**80.86**	18.09	11.81	65.48	94.20	1.45	**93.40**	1.04	**35.25**	51.95	36.17
Gemini 3.8 Flash	**95.56**	**0.00**	76.90	18.02	4.86	72.73	**98.55**	**0.00**	87.50	**0.00**	17.21	32.37	32.10
Qwen3.5-Plus	60.56	6.71	55.12	34.88	0.00	57.14	86.23	2.17	46.18	24.11	1.64	61.70	49.79
(2) Generation-and-verification frameworks (GPT-5.5 backbone)													
Direct (GPT-5.5, matched)	70.00	5.00	75.25	5.21	**22.92**	70.83	93.48	**0.00**	85.42	5.56	28.69	34.43	39.61
LLM4PLC	75.00	**0.00**	**77.23**	4.95	0.00	**43.75**	97.83	**0.00**	84.38	6.25	23.77	48.36	43.97
Agents4PLC	73.33	3.33	**77.23**	10.89	0.00	79.17	84.78	**0.00**	70.83	**0.00**	16.39	27.87	49.05
AutoPLC	**80.00**	**0.00**	75.25	4.95	12.50	75.00	**100.00**	**0.00**	87.50	7.29	**31.15**	**25.41**	41.44
SemaPLC	76.67	**0.00**	65.35	**0.99**	12.50	85.42	93.48	**0.00**	**94.79**	**0.00**	9.84	39.34	45.45

Direct-model rows pool repeated runs; matched direct is the GPT-5.5 baseline used for framework comparison.

5.2 Main Results

Table 4 reports SR and SVR by suite and difficulty together with ExecutionGap.

Plant-level performance. Figure 5 shows substantially lower SR on Hard tasks than on Easy and Medium tasks across both generation settings. When both suites are pooled by case count, direct GPT-5.5 achieves 82.70% and 84.09% SR on Easy and Medium tasks, respectively, compared with 25.10% on Hard tasks. Other models and frameworks show similar declines. These comparisons describe differences between task groups rather than isolating the effect of dependency scope.

Completion and safety. Higher SR is not necessarily accompanied by lower SVR. On Hard Material Handling tasks, Gemini 3.1 Pro achieves higher SR than GPT-5.5 (35.25% vs. 27.87%), but also higher SVR (51.95% vs. 32.24%). Thus, task completion alone does not capture observed safety violations, while a lower SVR does not necessarily indicate better completion. These results support reporting *Task Success* and *Safety Violation* separately.

Verification and overhead. The adapted generation-and-verification workflows show mixed changes across task groups. Relative to the matched GPT-5.5 baseline, AutoPLC improves both SR and SVR on Hard Material Handling tasks but worsens both on Hard Motion Control tasks (Table 4). On the shared 18-task subset, all evaluated frameworks use more model calls, tokens, and elapsed time than direct GPT-5.5 generation (Table 5). LLM4PLC and AutoPLC each use a median of four calls while revising the initial program for only 5 and 2 of the 18 tasks, respectively. Overall, higher measured generation overhead is not accompanied by consistent plant-level gains in this comparison. Paired SR comparisons and internal workflow-status outcomes are reported in Appendices H.2 and H.3, respectively.

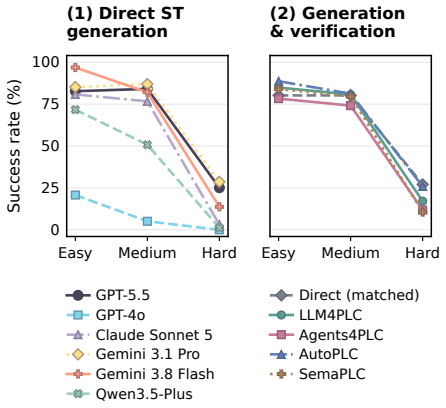

Figure 5: Suite-combined success rate by difficulty.

Table 5: Generation cost on the 18-task subset.

Model / Method	Calls	Revised	Tok. (k)	Time (s)
(1) Direct ST generation				
GPT-5.5	1	0/18	11.51	99.29
GPT-4o	1	0/18	4.45	5.90
Claude Sonnet 5	1	0/18	17.00	122.32
Gemini 3.1 Pro	1	0/18	19.17	118.09
Gemini 3.8 Flash	1	0/18	25.80	65.54
Qwen3.5-Plus	1	0/18	19.80	250.76
(2) Generation-and-verification frameworks				
LLM4PLC	4	5/18	48.28	321.13
Agents4PLC	5	10/18	73.11	452.56
AutoPLC	4	2/18	30.96	134.34
SemaPLC	18	10/18	523.88	230.90

Calls, tokens, and time are per-task medians. Revised is the number of tasks with at least one revision (out of 18).

5.3 Analysis

5.3.1 Execution Gap

The reported *Execution Gap* measures task failures or observed safety violations among cases assigned to programs passing the local submission-profile check, not external compilation or formal verification. Table 4 shows substantial gaps in both generation settings; Figure 6 shows direct-generation results. Since acceptance rates vary, comparisons require accepted/assigned counts. Gemini 3.8 Flash has lower *Execution Gap* than GPT-5.5 (32.10% vs. 37.88%), but fewer accepted cases (1,187/1,419 vs. 1,399/1,419) and fewer assigned cases achieving *Task Success* without an observed *Safety Violation* (806/1,419 vs. 869/1,419). Lower *Execution Gap* alone therefore does not imply better overall outcomes.

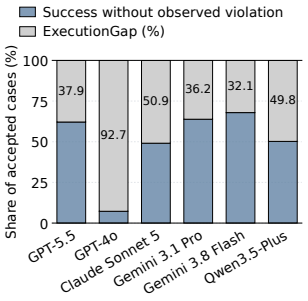

Figure 6: *Execution Gap*

5.3.2 Failure Analysis

We examine how task failures and safety violations manifest in cases entering plant execution after passing the common upstream check. The analysis includes successful executions with a violation but excludes generation and pre-execution failures, which provide no plant observations. Table 6 defines eight evaluator-recorded signatures grouped into completion protocol, safety constraints, and required behavior. The fixed mapping is multi-label: cases may activate multiple signatures, so percentages need not sum to 100%. Figure 7 reports prevalence across $4{,}339$ adverse cases pooled over models and methods ($3{,}615$ with failed-rule detail); matched direct is counted once.

Table 6: Definitions of evaluator-recorded failure signatures.

Signature	Recorded manifestation
Completion protocol	
Completion timing	No valid completion is established by the required deadline
Completion validity	Completion is declared before the required conditions are satisfied
Completion retention	A required completed state or completion signal is not maintained
Safety constraints	
Actuation constraints	Actuation conflicts with another command or violates an interlock
Spatial constraints	Execution violates a collision, clearance, or travel constraint
Resource constraints	Occupancy, capacity, holding, or support requirements are violated
Required behavior	
Goal fulfillment	The requested operation or required physical goal is not achieved
Process conformance	Required order, route, identity, direction, or quantity is incorrect

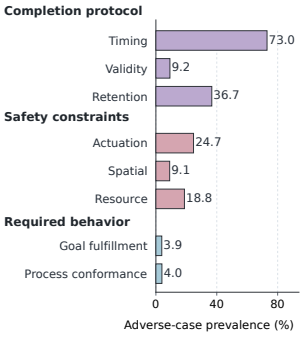

Figure 7: Failure-signature prevalence

Completion timing is the most frequent signature (73.0%), while completion retention affects 36.7% of adverse cases, reflecting failure to maintain a required completed state or completion signal. In one *Motion Control* case, a GPT-5.5 program asserts its completion output, but the plant state after that scan fails the required completion condition; the evaluator records premature completion. Actuation and resource constraint violations affect 24.7% and 18.8% of adverse cases, respectively.

The pooled percentages describe the composition of adverse cases, not overall failure rates, differences across models or methods, or root causes in the code. Appendix I provides the failure breakdown by stage, disaggregated results, counts, and mappings from rules to signatures.

6 Conclusion

We present PLCWorld as a common resource for reproducible closed-loop evaluation of LLM-generated ST programs, with 100 tasks, 473 registered task–condition pairs, and separate *Task Success* and *Safety Violation* judgments. Reference and alternative programs provide evidence for task feasibility and acceptance of tested valid implementations, while controlled counterexamples assess targeted violation detection. Evaluations of six LLMs and four generation-and-verification frameworks demonstrate distinct completion and safety outcomes across task groups.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript revision, including drafting and refining passages, and minor code edits. The LLMs evaluated in our experiments and their generation procedures are described in Section 5. The authors take responsibility for the final manuscript and code, including all AI-assisted content.

ETHICS STATEMENT

Errors in industrial control programs can damage equipment or endanger people. PLCWorld evaluates generated ST programs in simulation and reports *Task Success* and *Safety Violation* separately. Its safety metric covers specified constraints under evaluated conditions and does not constitute safety certification. Physical deployment requires independent engineering review and equipment-specific validation. The benchmark tasks instantiate abstracted industrial control relations using newly defined goals, I/O mappings, and parameters, without reproducing the source implementations. Confidential source programs and engineering documents are excluded from the planned public release. Four practitioners reviewed the public task specifications for industrial plausibility, information sufficiency, completion and safety clarity, and difficulty appropriateness. Reviewer names and employers are omitted; only broad experience ranges and areas of expertise are reported.

REPRODUCIBILITY STATEMENT

Section 4 and Appendix F describe benchmark construction and validation. Appendices B and C document task specifications and plant models. Section 5 and Appendix G provide the prompts, model configurations, framework adaptations, evaluation versions, and statistical and efficiency measurement procedures. The simulator, public task specifications, evaluator, submission interface, and baseline scaffold are available for anonymous review at `https://anonymous.4open.science/r/PLCWorld-6DAB/` and will be released publicly upon acceptance. Confidential industrial source materials are not included.

REFERENCES

Borja Fernández Adiego, Dániel Darvas, Enrique Blanco Viñuela, Jean-Charles Tournier, Simon Bliudze, Jan Olaf Blech, and Víctor Manuel González Suárez. Applying model checking to industrial-sized plc programs. *IEEE Transactions on Industrial Informatics*, 11(6):1400–1410, 2015.

Thiago Alves and Thomas Morris. Openplc: An iec 61,131–3 compliant open source industrial controller for cyber security research. *Computers & Security*, 78:364–379, 2018.

Anthropic. Claude Sonnet 5 system card. `https://www.anthropic.com/claude-sonnet-5-system-card`, 2026. Accessed September 21, 2026.

Autonomy Logic. OpenPLC Runtime v4. `https://github.com/Autonomy-Logic/openplc-runtime`, n.d. Official runtime documentation. Accessed September 11, 2026.

Hany Bassily, Rajat Sekhon, David E Butts, and John Wagner. A mechatronics educational laboratory–programmable logic controllers and material handling experiments. *Mechatronics*, 17(9):480–488, 2007.

beremiz. MATIEC, 2017. URL `https://github.com/beremiz/matiec`. Source code repository.

CODESYS Group. CODESYS Group, n.d. URL `https://www.codesys.com/`. Accessed September 21, 2026.

Mohamad Fakih, Rahul Dharmaji, Yasamin Moghaddas, Gustavo Quiros, Oluwatosin Ogundare, and Mohammad Abdullah Al Faruque. Llm4plc: Harnessing large language models for verifiable programming of plcs in industrial control systems. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, pp. 192–203, 2024.

Christopher A Field and Alan H Welsh. Bootstrapping clustered data. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 69(3):369–390, 2007.

Google DeepMind. Gemini 3.1 Pro model card. https://deepmind.google/models/model-cards/gemini-3-1-pro/, 2026a. Accessed September 21, 2026.

Google DeepMind. Gemini 3.8 Flash model card. https://deepmind.google/models/model-cards/gemini-3-8-flash/, 2026b. Accessed September 21, 2026.

Nan Huo, Xiaohan Xu, Jinyang Li, Per Jacobsson, Shipei Lin, Bowen Qin, Binyuan Hui, Xiaolong Li, Ge Qu, Shuzheng Si, et al. Bird-interact: Re-imagining text-to-sql evaluation via lens of dynamic interactions. In *International Conference on Learning Representations*, volume 2026, pp. 44285–44333, 2026.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024. doi: 10.48550/arXiv.2410.21276.

International Electrotechnical Commission. *IEC 61131-3:2013 Programmable Controllers – Part 3: Programming Languages*. International Electrotechnical Commission, 3.0 edition, 2013. URL https://webstore.iec.ch/en/publication/4552.

Heiko Koziolek, Thilo Braun, Virendra Ashiwal, Sofia Linsbauer, Marthe Ahlgreen Hansen, and Karoline Grotterud. Spec2control: Automating plc/dcs control-logic engineering from natural language requirements with llms-a multi-plant evaluation. In *Proceedings of the IEEE/ACM 48th International Conference on Software Engineering: Software Engineering in Practice*, pp. 566–577, 2026.

Yin Li, Liangwei Wang, Shiyuan Piao, Boo-Ho Yang, Ziyue Li, Wei Zeng, and Fugee Tsung. Mcoder: Streamlining motion control with llm-assisted code generation and rigorous verification. In *2025 IEEE 21st International Conference on Automation Science and Engineering (CASE)*, pp. 1597–1603. IEEE, 2025.

Zihan Liu, Ruinan Zeng, Dongxia Wang, Gengyun Peng, Xiaoxia Liu, Qiang Liu, Peiyu Liu, Wenhai Wang, and Jingyi Wang. Agents4plc: Automating closed-loop plc code generation and verification in industrial control systems using llm-based agents. *IEEE Transactions on Software Engineering*, 2026.

Mitsubishi Electric. Regarding simulation function using GX Works3. https://fa-faq.mitsubishielectric.com/faq/show/22327, 2023. Official FAQ 22327, updated February 15, 2023. Accessed September 11, 2026.

Mitsubishi Electric. 3d simulator MELSOFT Gemini. https://dl.mitsubishielectric.com/dl/fa/document/catalog/sol/3dsim/l08816eng/l08816engc.pdf, n.d. Official product catalog L(NA)08816ENG-C. Accessed September 11, 2026.

OpenAI. GPT-5.5 System Card, April 2026. URL https://openai.com/index/gpt-5-5-system-card/. Updated April 24, 2026.

Tolga Ovatman, Atakan Aral, Davut Polat, and Ali Osman Ünver. An overview of model checking practices on verification of plc software. *Software & Systems Modeling*, 15(4):937–960, 2016.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Michael Ruderman, Makoto Iwasaki, and Wen-Hua Chen. Motion-control techniques of today and tomorrow: a review and discussion of the challenges of controlled motion. 2020.

Robert G Sargent. Verification and validation of simulation models. *Journal of simulation*, 7(1): 12–24, 2013.

Siemens. *Virtual Commissioning of Machines with S7-PLCSIM Advanced, SIMIT and NX MCD: Getting Started*. Siemens AG, 2024. URL https://cache.industry.siemens.com/dl/files/943/109758943/att_1265915/v1/Manual_VirtualCommissioningOfMachines_GettingStarted_V1.0.1_EN.pdf. Application example 109758943, version 1.0.1.

Siemens AG. *SIMATIC S7-PLCSIM Advanced API Programming Manual*, November 2024. URL https://cache.industry.siemens.com/dl/files/690/109977690/att_1308929/v1/s7-plcsim_advanced_function_manual_API_en-US_en-US.pdf. V7.0, A5E53246567-AC.

Yanlun Tu, Huacan Wang, Ziyue Zhou, Jie Zhou, Ningyan Zhu, Ge Chen, Wangyi Chen, Tengfei Zhou, Yifan Zhou, Dasheng Yang, et al. Semaplc: A project-grounded, verification-gated agent harness for plc code generation. *arXiv preprint arXiv:2608.18565*, 2026.

Valeriy Vyatkin. Software engineering in industrial automation: State-of-the-art review. *IEEE Transactions on Industrial Informatics*, 9(3):1234–1249, 2013.

Tian Xia, Tianrun Gao, Wenhao Deng, Long Wei, Xiaowei Qian, Chenglei Yu, and Tailin Wu. Buildarena: A physics-aligned interactive benchmark of llms for engineering construction. *arXiv preprint arXiv:2510.16559*, 2025.

Donghao Yang, Aolang Wu, Tianyi Zhang, Li Zhang, Xiaoli Lian, Fang Liu, Yuming Ren, Jiaji Tian, and Xiaoyin Che. Autoplc: Generating vendor-aware structured text for programmable logic controllers. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 3593–3604. IEEE, 2025.

APPENDIX CONTENTS

A Limitations **14**

B Benchmark Contents and Task Specifications **14**
B.1 Benchmark Composition 14
B.2 Public Task Specification Format 15
B.3 Standard Execution Conditions 15
B.4 Representative Task 15

C Plant Model Details **16**
C.1 Operational Device Families 17
C.2 Transition Rules 19
C.3 Workpiece and Transferable-Fixture Relations 19
C.4 Modeling Scope 19

D Industrial Grounding and Task Construction **20**
D.1 Industrial Source Materials and Relation Abstraction 20
D.2 Synthetic Task Instantiation 20

E Evaluation Rules and Metric Definitions **21**
E.1 Execution Records and Temporal Conditions 21
E.2 Task Success and Safety Violation 21
E.3 Errors, Termination, and Metric Eligibility 21

F Benchmark and Runtime Validation **22**
F.1 Task Specification Validation 22
F.2 Evaluator Judgment Validation 25
F.3 Specification–Evaluator Alignment Validation 25
F.4 ST Runtime Validation 25

G Experimental Setup and Reproducibility **26**
G.1 Benchmark Version and Evaluation Units 26
G.2 Model Configurations and Input Protocol 26
G.3 Framework Adaptations and Resource Budgets 28
G.4 Acceptance Checks and Metric Accounting 29
G.5 Statistical Analysis and Efficiency Measurement 29

H Additional Experimental Results **29**
H.1 Execution Coverage and Joint Outcomes 30
H.2 Confidence Intervals and Generation Variability 31
H.3 Framework Evaluation Outcomes 32
H.4 Generation Efficiency 33

I Execution Gap and Failure Analysis **34**
I.1 Common Upstream Check and Execution Gap Counts 34
I.2 Rule-to-Signature Mapping 35
I.3 Outcome and Signature Distributions 35
I.4 Representative Evaluation Outcomes 36

A LIMITATIONS

Runtime comparisons cover selected constructs and programs, with residual cross-runtime differences (Appendix F.4). Broader runtime coverage and validation of plant responses against measurements from physical equipment remain future work. PLCWorld's tasks are grounded in PLC programs and engineering documentation from real-world manufacturing equipment. Limited source coverage constrains generalization across industrial settings, while source confidentiality restricts independent inspection of this grounding. Evaluation covers specified requirements under registered execution conditions, leaving unmodeled hazards and untested executions outside its scope. Reported *Execution Gap* is conditioned on the common upstream check for the supported ST profile and required I/O interface. Framework comparisons reflect our PLCWorld adaptations under method-specific resource budgets.

B BENCHMARK CONTENTS AND TASK SPECIFICATIONS

This appendix describes the composition of PLCWorld and the model-facing task specifications used for evaluation. We report statistics for the 100 benchmark tasks, document the information provided to the evaluated models, describe the standard execution conditions, and provide representative task objectives and plant configurations. These descriptive statistics are not used to assign task difficulty, which is defined by the scope of control dependencies in Section 3.2.

Table 7: Benchmark composition by suite and difficulty. I/O counts all distinct addresses declared in each task's public specification, including `BOOL`, `INT`, and `DINT` signals. Device counts denote distinct operational device families per task under the author-defined 17-family taxonomy.

Suite / Difficulty	#Tasks	#Cond.	I/O avg.	I/O range	Device avg.	Device range
Material / Easy	15	46	8.0	5–12	1.20	1–2
Material / Medium	20	96	17.4	11–28	2.85	1–7
Material / Hard	15	122	57.5	37–75	5.13	2–9
Motion / Easy	15	60	8.9	7–12	1.07	1–2
Motion / Medium	20	101	20.1	12–29	2.10	1–4
Motion / Hard	15	48	128.4	64–146	3.87	3–5
Total	100	473	37.9	5–146	2.68	1–9

B.1 BENCHMARK COMPOSITION

Table 7 summarizes the number of tasks, standard execution conditions, public I/O addresses, and operational device families in each suite–difficulty group. Condition counts are taken from the frozen evaluation manifest. I/O counts are computed from the distinct addresses declared in each public task specification.

For composition statistics, operational devices are classified using the author-defined taxonomy of 17 families documented in Appendix C.1. Multiple instances of the same family within a task are counted once. The resulting counts characterize device-family diversity rather than the number of physical devices or controlled axes. These composition statistics are descriptive and are not used to assign difficulty.

Requirement-family distribution. Table 8 provides the suite–difficulty breakdown of the aggregate requirement-family counts reported in Figure 4. A family is assigned only when the public task specification imposes the corresponding control obligation on the generated program; device presence alone is insufficient. Assignments are non-exclusive: each task is counted at most once per family but may belong to multiple families. These annotations characterize task content and are separate from difficulty, which is defined by the scope of control dependencies.

Table 8: Distribution of control requirement families by suite and difficulty. Counts are descriptive and non-exclusive; a task may contribute to multiple rows.

	Material Handling			Motion Control			
Requirement family	**Easy**	**Medium**	**Hard**	**Easy**	**Medium**	**Hard**	**Total**
Positioning and stabilization	8	19	14	14	20	15	90
Synchronization and compensation	0	1	1	4	4	9	19
Access and clearance	0	12	14	2	17	15	60
Gripping and support	5	13	14	1	4	1	38
Quantity and inventory	4	6	8	0	0	0	18
Identification and routing	4	7	14	0	0	15	40
Data and reference validity	4	7	10	2	3	3	29
Occupancy and capacity	1	2	15	0	0	15	33
Processing and separation	0	1	14	0	0	0	15

B.2 Public Task Specification Format

Each PLCWorld task is distributed as a model-facing specification that defines the required external behavior without prescribing an internal program structure. Within the supported ST profile, a submission may choose its internal variables, branching logic, timers, and state-machine organization, provided that its execution satisfies the published plant-level requirements.

The specification provides the task objective, completion deadline, evaluation endpoint or termination condition, public I/O interface and signal semantics, initial plant state, permitted operating conditions, device-response semantics, completion requirements, and safety constraints. Where relevant, it also identifies intermediate events and precedence relations required for completion. Device descriptions state how task-relevant plant state changes in response to outputs and how the resulting state is exposed through sensor inputs. Table 9 summarizes the principal information contained in a public task specification.

The model-facing input excludes reference ST programs and evaluator source code. It provides the task requirements and permitted operating ranges, but not the complete configurations of individual registered execution conditions. Submitted programs are evaluated against the published requirements without requiring the reference program's internal control structure.

B.3 Standard Execution Conditions

A task may be evaluated under multiple execution conditions. Each condition instantiates concrete values for quantities already allowed by the public task specification, while leaving the task objective, public I/O interface, completion requirements, and safety constraints unchanged. Across PLCWorld, the 100 tasks define 473 standard task–condition pairs (Table 7).

Execution conditions may vary the initial plant state, device response or timing values, and external inputs such as requests or start conditions when these quantities are variable in the task specification. Sensor feedback is not supplied as a fixed trace: during execution it is generated from the plant state reached under the selected condition and the submitted program's preceding outputs.

The standard conditions are frozen as part of the benchmark release and are also used during benchmark construction and validation.

Table 10 summarizes what remains fixed and what may vary across conditions of the same task.

B.4 Representative Task

Table 11 summarizes representative task objectives across both suites and all three difficulty levels, and Figure 8 shows the corresponding plant configurations. These examples illustrate the progression from a basic control function, to dependencies between functions of one job, to coordination between independent jobs. The complete public specification for each task additionally contains

Table 9: Principal components of a PLCWorld public task specification. The specification defines required observable behavior while allowing implementation choices within the supported ST profile.

Component	Information provided	Role in evaluation
Task objective	Required functional outcome and, where applicable, the requested target, item, quantity, destination, or operating mode.	Defines the plant-level result the submitted program must achieve.
Timing and completion	PLC scan period, completion deadline, evaluation endpoint or termination condition, and any required completion signal or post-completion retention.	Defines scan timing, completion validity, required result retention, and when evaluation ends.
Public I/O interface	Signal names, addresses, types, and input/output semantics.	Defines the interface through which the ST program observes and controls the task environment.
Initial plant state	Initial device states, positions, workpiece or fixture relations, and relevant request or process state.	Defines the starting state for each execution.
Operating conditions	Permitted ranges or alternatives for device responses, initial conditions, and external inputs.	Defines the range within which evaluation conditions may vary.
Device-response semantics	Task-relevant effects of activating and deactivating outputs, including timing and transition conditions.	Defines how program outputs can change device, process, workpiece, or fixture state.
Sensor semantics	Meaning of feedback inputs such as position, operation completion, holding, occupancy, validity, or process state.	Defines what evidence the program can use to determine the effect of its previous commands.
Required events and relations	Intermediate events, precedence requirements, handover conditions, or state dependencies required for valid completion.	Distinguishes valid task execution from reaching a similar final state through an invalid sequence.
Safety constraints	Prohibited actions or states and task-specific interlock requirements.	Defines the conditions monitored for *Safety Violation*.

Table 10: Relationship between a task specification and its execution conditions. Condition-specific values remain within the operating ranges published for the task.

Fixed across conditions	Task objective; public I/O interface and semantics; device-response rules; completion requirements and deadline rules; safety constraints.
May vary by condition	Initial plant state, permitted device-response or timing values, and task-defined external inputs or request parameters.

the I/O interface, initial state, operating conditions, device-response semantics, completion requirements, and safety constraints described in Appendix B.2.

The examples are illustrative rather than additional task definitions. Evaluation always uses the complete frozen public specification associated with each task ID.

C PLANT MODEL DETAILS

This appendix documents the executable plant abstractions used by PLCWorld. The plant model is task-specific: each task instantiates only the device, process, workpiece, fixture, and numerical states needed to realize its public specification and evaluate task completion and safety. The sections below describe the operational device-family taxonomy used for corpus statistics, the transition semantics

Table 11: Representative PLCWorld task objectives across suites and difficulty levels. Full model-facing specifications are distributed with the benchmark.

Task	Suite / Difficulty	Public objective
MOT-E14	Motion / Easy	Suppress the sway of an empty hoist hook and bring the trolley to rest at its current work position.
MOT-M16	Motion / Medium	Position the table and carriage for tool access, maintain their alignment throughout the requested tool cycle, confirm completion of that cycle, and retract the carriage.
MOT-H03	Motion / Hard	Coordinate two independent setup requests through lift stations, a shared shuttle, moving docking, and rotary synchronization, satisfying relative-motion constraints and returning each fixture.
MAT-E06	Material / Easy	Discharge one product from the already aligned pocket and retract the pusher, while preserving the products in all other pockets.
MAT-M14	Material / Medium	Prepare a single product, clear the feeder, and time its transfer into a continuously moving receiving pocket, while preserving the reserve product.
MAT-H05	Material / Hard	Coordinate assembly, inspection, rework, and outfeed on a shared indexing table, preserving body–insert pairing and current inspection results, clearing all tools before indexing, and reinspecting reworked products before release.

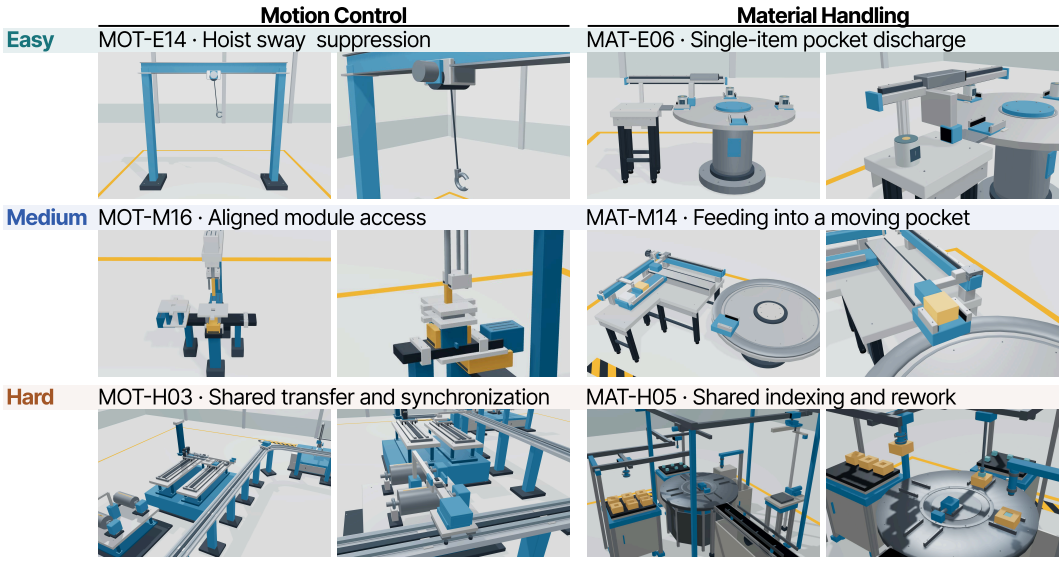

Figure 8: Plant configurations for the representative tasks in Table 11, organized by task suite and difficulty level.

used during execution, the representation of workpieces and transferable fixtures, and the scope of the modeled physics.

C.1 Operational Device Families

Table 12 classifies PLCWorld's operational devices into 17 author-defined families. The same taxonomy underlies the device-family statistics in Table 7, counting each family at most once per task regardless of the number of instances. This convention describes benchmark composition, not an industry-standard taxonomy or the simulator's software architecture.

The counts include devices that participate in task operation, including externally driven and automatically operated devices. Ordinary sensors, fixed supporting geometry, visual-only components, inactive configuration branches, and internal components subsumed by a higher-level device are excluded. Commands and feedback in the table are illustrative semantic summaries; actual I/O names

and addresses, response parameters, and admissible transitions follow each task's public specification.

Table 12: Author-defined operational device families in PLCWorld. Commands and feedback are semantic summaries rather than universal I/O names. Each family is counted at most once per task.

Family	Typical command	Typical feedback / state	Modeled role
`linear_axis`	enable, extend/retract, target, rate	position, velocity, endpoint, settled	Positions a tool, fixture, counterweight, pusher, slide, or other linearly moving mechanism.
`rotary_axis`	enable, index, target angle, rate	angle, angular velocity, at-station, settled	Orients or indexes a shaft, disk, table, tilt head, or slew mechanism.
`lift_hoist`	raise/lower, target height, enable	height, velocity, brake or settled state	Changes the elevation of a supported load, tool, fixture, or suspended item.
`transfer_module`	move, insert/retract, dock, couple	position, inserted, docked, coupled, occupied	Transfers a workpiece or fixture between supports, stations, or process modules.
`manipulator`	pose or taught-motion request, sequence	at-pose, busy, motion complete	Moves a tool or held item through an articulated or explicitly identified Cartesian robot.
`mobile_robot`	motion enable, velocity, steering or target	pose, heading, velocity, arrival	Represents a mobile wheeled platform and its task-relevant motion.
`gripper`	close, open, grip, release	open, closed, holding	Establishes or clears mechanical ownership of a workpiece.
`vacuum`	vacuum on/off, acquire, vent	vacuum present, holding, released	Establishes or clears suction-based holding.
`clamp_lock`	clamp/unclamp, lock/unlock	clamped, open, locked, holding	Secures a workpiece, fixture, web, or docking relation in place.
`conveyor`	run, stop, direction, speed	occupied, position, boundary crossing	Transports material on an explicitly modeled belt or powered roller while maintaining support.
`fluid_delivery`	pump, fill, drain, valve selection	level, delivered quantity, line or valve state	Transfers a modeled liquid quantity between a supply, line, and container.
`web_handling`	feed, wind, unwind, nip	length, feed position, storage, reel state	Supplies, stores, or takes up continuous web material.
`cutting_press`	cut, press, punch, return	busy, returned, cut or press complete	Separates, punches, or mechanically forms material through a modeled operation.
`process_module`	start, cycle, machine, couple	ready, busy, done, process result	Performs a task-specific machining, assembly, coupling, or surface operation.
`inspection`	measure, inspect, weigh	busy, result valid, measured value, identity	Produces a measurement or classification result associated with a workpiece.
`gate_door`	open, close	open, closed, clear, busy	Controls access or clearance to a station or protected operating region.
`packaging_applicator`	apply, place, cap, tape	busy, applied, complete	Adds or applies a modeled packaging component such as a cap or tape.

C.2 Transition Rules

Device states evolve according to task-specific response dynamics, timing ranges, and transition preconditions rather than being established instantaneously by commands. For example, a gripping command establishes holding only after the closing response has completed while the workpiece is in an admissible acquisition configuration. Depending on the task, an inadmissible command may be blocked, have no physical effect, or be recorded as a safety violation.

The evaluator assesses *Task Success* from the required goal plant state and event history, subject to the specified deadline and hold conditions. When a completion signal is required, the plant state after applying it for one scan must satisfy the completion conditions. Command or completion bits alone are insufficient.

C.3 Workpiece and Transferable-Fixture Relations

For tasks involving material or transferable fixtures, the plant model maintains the subset of state properties required by the task. These properties are task-specific rather than fields that must appear in every plant model. They commonly include:

- **Location**: the current position, station, slot, pocket, layer, or region occupied by a workpiece or fixture.
- **Owner / holder**: the tool, carrier, support, station, or process module that currently has custody of the item. Ownership changes only through a modeled acquisition, release, placement, or handover transition.
- **Support status**: whether the item is held, supported, in transfer, released onto a valid receiver, or unsupported when that distinction is relevant to completion or safety.
- **Identity and request association**: a persistent identifier, token, order, lot, or destination used when an item must remain associated with its originating request or result.
- **Quantity and arrangement**: task-specific counts, lengths, material amounts, slots, layers, or sequence positions used for batching, stacking, kitting, storage, or dispatch.
- **Process and inspection state**: task-specific machining, cutting, assembly, measurement, acceptance, or classification results associated with an item.
- **Transition history**: acquisition, release, handover, acceptance, processing, or acknowledgement events needed to distinguish a valid execution from the same final state reached through an invalid sequence.

Motion Control tasks need not contain a conventional material workpiece. In such tasks, the relevant transferred entity may instead be a fixture, carrier, jig, or request-associated load, and some tasks evaluate only device motion and feedback states.

C.4 Modeling Scope

PLCWorld uses task-specific plant models combining discrete events with task-relevant numerical state rather than a general-purpose physics simulator. The plant models represent device responses, timing, sensor updates, command preconditions, and workpiece or fixture transitions required to evaluate task completion and safety. Depending on the task, the modeled state may also include quantities such as position, velocity, occupancy, material amount, mass, balance relations, and collision or clearance geometry.

Contact forces, friction, deformation, process quality, and multi-body dynamics are modeled only to the extent explicitly represented through published state variables, transition rules, geometric checks, or numerical relations. Each plant model is an executable abstraction of its task specification, not a complete physical model of the corresponding equipment.

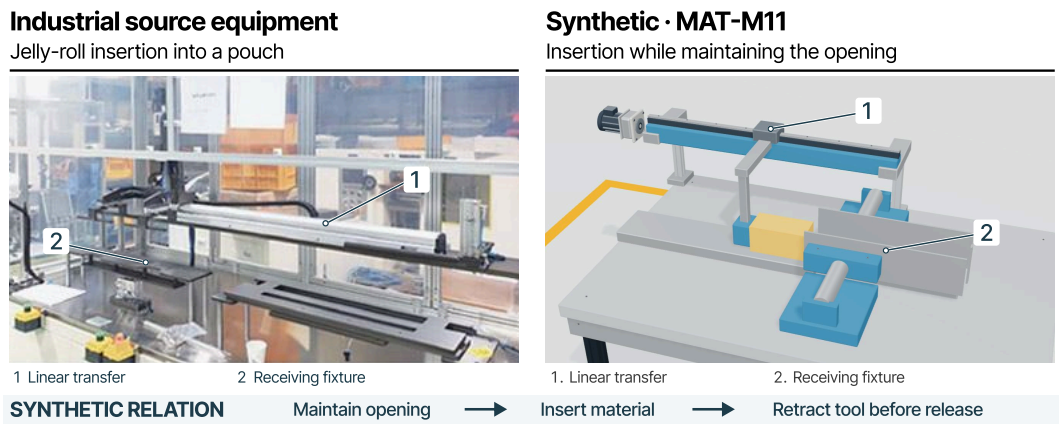

Figure 9: Industrial source equipment and synthetic PLCWorld task MAT-M11. Matching numbers indicate corresponding device roles; the bottom sequence summarizes the synthetic control relations.

D Industrial Grounding and Task Construction

D.1 Industrial Source Materials and Relation Abstraction

We cross-referenced PLC programs with the engineering documents listed in Table 13 to interpret commands, feedback signals, and device states. No single artifact was treated as a complete task specification.

We abstracted control relations specifying an action, its observable state or feedback, prerequisite device or workpiece states, and states to maintain during subsequent operations. Source-specific tags, vendor functions, numerical setpoints, and equipment layouts were excluded from the public tasks. The source PLC programs and engineering documents are not included in the benchmark release.

Table 13: Roles of the industrial source materials in task construction.

Source material	Information consulted	Use in task construction
PLC programs	Actuation conditions, retained states, interlocks, and request–completion logic	Identification of command–feedback and sequencing relations
I/O lists and electrical drawings	Signal direction, device association, and feedback source	Interpretation of commanded and observed plant states
Equipment manuals	Device operation, completion indication, and operating constraints	Interpretation of task-relevant device and sensor behavior
Operating-sequence and engineering documents	Operation order, readiness conditions, and handover context	Identification of precedence, support, clearance, and shared-resource relations

D.2 Synthetic Task Instantiation

For each task, we selected a functional objective informed by industrial equipment documentation and instantiated the associated control relations in a new synthetic plant configuration. We specified the devices and workpieces, initial plant state, public I/O interface and semantics, and plant-model parameters.

Figure 9 compares industrial source equipment with a synthetic PLCWorld task. Matching annotations identify corresponding device roles, while the accompanying sequence summarizes the control relations specified for the synthetic task. This comparison illustrates task abstraction, rather than equivalence in physical layout, control implementation, or dynamics.

Each task specification defines the required event ordering and the plant states that must be maintained during subsequent operations. Completion requirements and safety constraints are expressed over plant states and events, without prescribing the program's internal implementation. Registered execution conditions assign concrete values within the task's permitted operating ranges (Appendix B.3).

E Evaluation Rules and Metric Definitions

E.1 Execution Records and Temporal Conditions

PLCWorld evaluates program–plant interaction using execution records. In each completed scan, the program reads the current input image and commits its outputs; the plant then updates its device and workpiece states. The record contains the inputs, committed outputs, updated plant state, and events. The evaluator also derives task-defined events from recorded states and outputs. Plant-level judgments do not depend on internal program variables or presumed control strategy.

State conditions describe plant states and, where required, committed outputs. Events mark occurrences such as acquisition, arrival, or handover. Precedence relations specify the required ordering of states or events; hold relations require a state to persist over a specified interval. These conditions and relations implement the public task requirements. Completion claims are checked against the plant state after the scan's outputs take effect and the event history through that scan. Issuing a command or claim alone does not establish the corresponding physical result.

E.2 Task Success and Safety Violation

For a metric-eligible case e, *Task Success* $U_e = 1$ only if all functional requirements are satisfied under its assigned execution condition. These include the goal state, required events and ordering, and any holding or retention requirements. The goal state and any required valid completion declaration must be established by the task-specific deadline; otherwise, $U_e = 0$, even if observation continues. A premature declaration does not establish completion; its further consequences follow the task specification. Loss of a required retained state or signal invalidates success.

For an eligible case entering plant execution, *Safety Violation* $V_e = 1$ if at least one specified safety constraint is breached during observation. Monitoring covers the observed execution, including any required period after a completion claim. Once observed, a violation remains counted even if the plant recovers. *Task Success* and *Safety Violation* are judged separately, although some requirements affect both. Functional failure alone is not a safety violation: success with a violation and failure without an observed violation are both possible. For interrupted executions, $V_e = 0$ means only that no violation was observed before interruption.

E.3 Errors, Termination, and Metric Eligibility

Cases with missing or unusable submissions, submission-profile violations, parsing failures, unsupported constructs attributable to the submitted program, or program-attributable execution errors remain eligible for SR with $U_e = 0$. Cases that never enter plant execution have no value of V_e and are excluded from SVR. For program-attributable interruptions after plant entry, the evaluator retains the observed execution prefix and any recorded violation; $U_e = 0$.

Normal termination follows the task-specific evaluation endpoint, which may be a deadline, a specified event, or the end of a required post-declaration observation period. A completion claim does not override this rule. Records identify plant entry, completion of the required observation, and any interruption point.

Infrastructure errors are excluded from all metric aggregation and reported separately. They are not converted into task failures or executions without an observed violation.

SR includes all metric-eligible assigned cases, whereas SVR includes only metric-eligible cases that entered plant execution. For an upstream check k, the *Execution Gap* denominator contains only metric-eligible cases whose exact submitted ST programs passed that check. Its numerator counts cases within this set that exhibit task failure or an observed safety violation, with each case counted

once. A metric with an empty denominator is reported as N/A. The common upstream check used in the main comparison covers the supported submission profile and required I/O interface, not external compilation or formal verification.

F Benchmark and Runtime Validation

This appendix details the validation procedures and results summarized in Table 3, covering task specifications, evaluator judgments, specification–evaluator alignment, and ST runtime behavior.

F.1 Task Specification Validation

Practitioner review complements executable checks by assessing whether the assumptions and control relations described in a task are reasonable for its intended industrial context (Sargent, 2013). We conducted a practitioner review to examine whether PLCWorld's public task specifications are industrially plausible and sufficiently specified for implementation. The review focused on properties that cannot be established solely through reference-program execution or evaluator testing, including industrial plausibility, information sufficiency, specification clarity, and the practical appropriateness of the difficulty levels.

F.1.1 Practitioner Background

The review involved four practitioners with professional experience in PLC programming, equipment control, and industrial automation. To preserve anonymity, we report broad experience ranges and areas of expertise rather than names, employers, or specific industrial projects.

Table 14 summarizes the reviewers' backgrounds. Each reviewer had at least five years of professional experience, with complementary expertise spanning PLC and control engineering, automation system integration, commissioning, and equipment control.

Table 14: Background of practitioners participating in the practitioner review. Names and employers are omitted to preserve anonymity.

Reviewer	Experience	Primary role	Relevant expertise
E1	5–10 years	PLC / Controls Engineer	PLC programming, sequencing, interlocks
E2	5–10 years	Automation / System Integration Engineer	Material handling, system integration, industrial networks
E3	10+ years	Commissioning Engineer	Commissioning, I/O checkout, troubleshooting, motion control
E4	10+ years	Equipment / Control Engineer	Servo and pneumatic systems, equipment operation

F.1.2 Review Scope and Procedure

Reviewers examined the public, model-facing task specifications; reference ST implementations and evaluator rules were not included in the review materials. The 100 tasks were organized into six suite–difficulty groups, with 15 Easy, 20 Medium, and 15 Hard tasks per suite. Each reviewer examined all specifications in each group, provided assessments for the group, and recorded concerns about individual tasks using the corresponding task IDs.

The review covered four aspects:

1. **Industrial plausibility**: whether the task objective, device configuration, and control relations are reasonable in an industrial automation setting;
2. **Information sufficiency and clarity**: whether the specification provides sufficient information about device behavior, initial conditions, I/O semantics, and command–feedback relations to implement the required PLC logic;

3. **Completion and safety clarity**: whether completion conditions, control dependencies, interlocks, and prohibited actions are stated clearly enough to determine correct behavior; and
4. **Difficulty appropriateness**: whether the assigned level reflects the scope of control dependencies required by the task.

For the first three aspects, the available ratings were *Appropriate*, *Minor Revision Required*, *Major Revision Required*, and *Unable to Judge*. For difficulty, reviewers assessed whether the assigned level was consistent with the stated dependency scope and recorded concerns about the assignment or its justification.

Reviewers could also record task-specific concerns involving industrial plausibility, missing implementation information, I/O or device behavior, completion or safety requirements, implementation restrictions, or difficulty assignment. A task could be associated with multiple issue types. These comments were analyzed separately from the assessments of each group.

F.1.3 Review Results by Group

Table 15 summarizes the assessments of the six suite–difficulty groups. Across the 24 assessments for each criterion, information sufficiency and completion and safety clarity were rated *Appropriate* in all cases. Industrial plausibility received 16 *Appropriate* and 8 *Minor Revision Required* ratings, primarily reflecting requests for clearer operational justification of selected synthetic tasks. Difficulty feedback called for clearer descriptions of dependency scope in two groups. These ratings summarize each group as a whole; concerns about individual specifications were recorded separately.

Table 15: Review assessments by suite and difficulty.

Suite	Difficulty	Industrial plausibility	Information sufficiency	Completion and safety clarity	Difficulty feedback
Motion Control	Easy	A: 3, MR: 1	A: 4	A: 4	Appropriate
	Medium	A: 2, MR: 2	A: 4	A: 4	Clarify dependency scope
	Hard	A: 2, MR: 2	A: 4	A: 4	Appropriate
Material Handling	Easy	A: 3, MR: 1	A: 4	A: 4	Appropriate
	Medium	A: 3, MR: 1	A: 4	A: 4	Appropriate
	Hard	A: 3, MR: 1	A: 4	A: 4	Clarify dependency scope

A: Appropriate; MR: Minor Revision Required. Numbers indicate reviewer counts. The difficulty column reports qualitative feedback rather than reviewer counts.

F.1.4 Task-Specific Concerns and Revisions

Reviewer comments identified opportunities to clarify operational rationale, completion requirements, and the dependency scope underlying difficulty assignments. Other comments concerned wording that could impose an unnecessary action order when alternative sequences satisfy the same completion and safety requirements. Table 16 summarizes the issue categories and the corresponding specification revisions.

Table 17 presents representative reviewer concerns, the specification revisions made in response, and their effects on the stated requirements.

F.1.5 Feedback on Difficulty Levels

Reviewers also assessed PLCWorld's dependency-based difficulty definition independently of individual task wording. The benchmark defines Easy tasks by dependencies contained within one basic function, Medium tasks by dependencies between functions of one job, and Hard tasks by dependencies between independent jobs sharing resources, space, or plant state.

Table 16: Task-specific concerns identified by reviewers and corresponding revisions. Only categories with recorded concerns are shown.

Issue type	# Tasks	Revision
Industrial plausibility	10	Clarified the operational purpose and industrial rationale of the affected tasks.
Completion condition	3	Specified the required final plant state, completion signal, and retention period where applicable.
Implementation restriction	4	Made alternative action orders explicit where they satisfy the same completion and safety requirements.
Difficulty rationale	7	Clarified the functional dependencies and shared-resource interactions supporting the existing difficulty assignments.

Counts refer to distinct tasks within each category; a task may appear in multiple rows. Difficulty revisions clarified the rationale without changing the assigned levels.

Table 17: Representative task specification revisions following practitioner review.

Task	Reviewer concern	Revision	Effect on specification
MOT-M02	The prohibition before request acceptance did not explicitly cover an enabled axis whose target equaled its current position.	Specified that no commanded axis may be enabled before acceptance of the `X0` request, even when its target would produce no movement.	Makes the start interlock explicitly apply to axis enabling, not only to actual movement.
MOT-M17	The return wording could imply a fixed axis order even when another safe return sequence was possible.	Specified when the feeder must remain retracted and explicitly permitted overlapping return movements when the protected geometry remains clear.	Preserves the required outbound dependency while allowing alternative safe return sequences.
MAT-H04	Whether completion required physical receipt of the packages and return of the handling devices was unclear.	Specified receipt and acknowledgement of all four packages, return of the empty shuttle, pickup head, and pusher, and retention of the completed state for 0.5 s.	Makes physical receipt, device return, and retention explicit parts of the completion requirements.
MAT-H13	The effect of an exit request with an empty gripper and the conditions for transporting a held bundle needed clarification.	Clarified that an exit request with an empty gripper moves nothing, while transferring a held bundle requires a free exit, a released wrapping clamp, and separated film.	Distinguishes an ineffective request from a valid bundle transfer and makes the required interlocks explicit.

Three of the four reviewers considered this distinction generally appropriate for representing differences in PLC control complexity. The remaining reviewer considered it acceptable provided that the explanations of dependency scope were clarified in response to the feedback.

Reviewer comments identified seven tasks whose descriptions required clearer justification of their assigned levels. We revised these explanations to make the relevant functional dependencies and coordination requirements explicit. No tasks were reassigned: each suite retained 15 Easy, 20 Medium, and 15 Hard tasks, and the experiments used these unchanged difficulty assignments.

We treat this assessment as evidence that the proposed levels are meaningful from a practitioner perspective, rather than as proof of an objective or universal ordering of PLC programming difficulty.

F.1.6 Specification Revision and Finalization

We revised 18 task specifications to address the reviewer concerns summarized above. The revisions clarified operational rationale, completion requirements, permitted action sequences, and the explanations supporting difficulty assignments. Revised specifications were returned to the reviewers for

follow-up. The benchmark version used in the experiments incorporated these revisions and was frozen after the review-and-revision process.

F.2 Evaluator Judgment Validation

F.2.1 Feasibility and Valid Alternatives

For each task, we run its reference ST program under every registered execution condition. All 100 reference programs achieve *Task Success* without an observed *Safety Violation* across the 473 task–condition pairs. This establishes a feasible implementation for each registered task and condition, but does not make the reference program's internal control sequence part of the acceptance criterion.

We also run 72 author-written alternative programs covering 67 tasks. These programs vary permitted control choices, including preparation order or scheduling where the specification allows it. All 378 corresponding executions achieve *Task Success* without an observed *Safety Violation*. This test checks that the evaluator accepts known valid implementations other than the reference; it does not establish acceptance of every possible valid program.

F.2.2 Targeted Violation Detection

We execute 542 controlled counterexample programs designed to violate specified completion, ordering, retention, command, or safety requirements. Each program has a prespecified evaluator rule to test, and all 542 activate their designated rule in at least one registered execution condition. These programs produce 739 executions across registered conditions, of which 723 are rejected. The remaining 16 executions occur under conditions in which those programs do not produce the targeted violation; the same programs activate their designated rules under other registered conditions. We therefore assess targeted detection per program while retaining all execution outcomes in the validation records.

F.3 Specification–Evaluator Alignment Validation

We check the public task inputs against the selected execution paths for all 100 tasks. The checks cover rendered input and source hashes, scan configuration, I/O addresses, types and directions, and submission entry points. All 100 tasks pass these checks. A separate comparison of 461 published numerical and operating-range items with the registered conditions finds no unresolved mismatch.

These checks are complemented by a bidirectional semantic audit: public obligations are compared with executable judgment conditions, and judgment conditions are checked for a basis in the public task specification. Identified discrepancies are classified and corrected. Following corrections, we rerun the full reference, alternative, and counterexample sets, obtaining the results reported above. Affected frozen model submissions are replayed without new model generation. These tests provide evidence for the audited obligations and registered conditions; they are not an exhaustive proof over arbitrary ST programs or continuous plant states.

F.4 ST Runtime Validation

Repeatability and diagnostic comparisons. Using the frozen evaluator, we rerun the reference suite in two independent processes. Trace hashes and judgments match across all 473 registered conditions, supporting repeatability under the tested configuration.

Three diagnostic ST programs are compared between PLCWorld's ST runtime and a pinned MATIEC/GCC toolchain (beremiz, 2017), using identical input images and virtual scan progression. All 1,320 observations over 360 scans agree. The diagnostics exercise retained state, `CASE`, loops, selected arithmetic, `R_TRIG`, and `TON`. Siemens S7-PLCSIM Advanced (Siemens AG, 2024) tests examine `TON` scan boundaries; an identified discrepancy was corrected and rechecked.

Comparison using generated programs. We use one admitted generation per task from each of six direct-generation models, yielding 600 unchanged submissions. Among those accepted by the pinned MATIEC compiler, we compare 268 programs across 86 tasks under all registered conditions (1,199 paired cases).

Both runtimes execute identical source under the same condition, with reset program memory and plant state and the same frozen plant and evaluator. Each run evolves independently through its own outputs and sensor feedback.

Complete scan-level records agree in $1{,}162/1{,}199$ cases, *Task Success* judgments in $1{,}195/1{,}199$, and *Safety Violation* judgments in all $1{,}199$. The 37 trace differences comprise 33 cases with matching final judgments and four with differing *Task Success* judgments.

All four *Task Success* differences involve a Qwen3.5-Plus program for task MOT-E08 (Motion Control, Easy) that calls the same `TON` instance twice per scan. In these cases, PLCWorld records *Task Success* false and MATIEC/GCC true; neither records a *Safety Violation*. Another Qwen3.5-Plus program for task MOT-H09 (Motion Control, Hard) contributes three trace differences with matching judgments.

The repeated-call difference is distinct from the corrected Siemens `TON` scan-boundary discrepancy. The comparisons cover the included programs and tested execution paths, not full ST-profile equivalence or independent validation of the shared plant and evaluator.

G Experimental Setup and Reproducibility

This appendix documents the prompts, model configurations, framework adaptations, and measurement protocols used in our experiments.

G.1 Benchmark Version and Evaluation Units

We use public model inputs v6 and evaluator v5.2.6. Each repetition evaluates one ST program per task across 100 tasks and 473 task–condition cases. Programs remain unchanged across their assigned conditions, with program memory and plant state reset before each case. Only complete repetitions without unresolved infrastructure errors enter full-benchmark comparisons. Earlier-version artifacts require documented replay confirming equivalent task-success and safety outcomes. We retain versioned inputs, submitted programs, and case-level evaluation records linked by source hashes.

G.2 Model Configurations and Input Protocol

Direct ST generation. Each direct-generation model receives the same system prompt and task-specific user prompt in three independent single-turn generations per task. Reference ST programs, evaluator source code, and complete configurations of registered execution conditions are withheld. Models have no access to tools, execution feedback, or previous attempts. All outputs are evaluated verbatim, without repair or best-of selection, yielding 300 programs and 1,419 evaluation cases per completed model.

System prompt

You write a PLC control program for the public task specification supplied in the user message.

Submission and I/O

- Return exactly one complete IEC 61131-3 Structured Text PROGRAM named Main, ending with `END_PROGRAM`. Return source only, without Markdown fences or explanatory prose.
- Declare every listed I/O using its exact address identifier and type. The task's single I/O table defines direction, access, and meaning. Input signals are read-only: do not assign them, use them as FOR counters, or connect function-block outputs to them. Only declared output signals are writable.

`[...]`

Scan and completion semantics

- At the start of a scan, external inputs and preceding device feedback are sampled. ST executes on that input image. Its outputs are then applied while device/workpiece states advance. Completion is checked after that scan's state updates, using the actual state, outputs and relevant history. Use the

System prompt (continued)

task's scan period, dynamics, input scaling and operating limits; an output command is not proof that its physical action has finished.

[...]

User prompt **MAT-E01**

MAT-E01 — Product Acquisition with Gripper

Objective

Acquire the single supported product with the parallel gripper.

[...]

I/O Interface

Address	Type	Direction	Meaning
X10	BOOL	input	Start request; one rising edge.
D10	INT	input	One product; D10=1.
Y2	BOOL	output	Completion claim, held until evaluation endpoint.
Y20	BOOL	output	grip: actuator command.
X20	BOOL	input	holding: physical observation.

Fixed I/O Declaration Skeleton

[...]

```
PROGRAM Main
VAR
    X10 : BOOL;
    D10 : INT;
    Y2 : BOOL;
    Y20 : BOOL;
    X20 : BOOL;
    (* Add internal variables here. *)
END_VAR
(* Implement the control logic here. *)
END_PROGRAM
```

[...]

Safety Constraints

- Do not initiate the requested operation before Start.

Completion Conditions

- The product belongs to the tool, the gripper is fully closed, and grip remains ON.
- Retain the result and Y2 through the remaining execution budget.

Task-specific Operating Conditions

- Gripper closing/holding and opening/release each respond in 0.10–0.20 s.
- The evaluation window lasts 1600 scans.
- Complete by scan 1550; at least 50 scans of result holding are checked.
- The PLC scan period is 10 ms.

Prompt excerpts; [...] marks omitted text. Formatting is adapted for readability.

Qwen3.5-Plus uses synchronous OpenRouter requests at concurrency eight; the other direct models use provider Batch APIs. Token limits and reasoning-token accounting differ across models, so inference compute is not matched. All frameworks use the GPT-5.5 configuration in Table 18.

Table 18: Model configurations for the completed direct-generation runs. "Default" indicates that the corresponding parameter was omitted from the request. Output limits are reported in tokens.

Model identifier	Reasoning / thinking	Temperature	Max. output tokens
`gpt-5.5-2026-04-23`	Medium	Default	32,768
`gpt-4o-2024-08-06`	—	Default	8,192
`claude-sonnet-5`	Default (adaptive)	Default	65,536
`gemini-3.1-pro-preview`	Default (high)	Default	65,536
`gemini-3.8-flash`	Medium	Default	32,768
`qwen/qwen3.5-plus-20260420`	Default (thinking)	Default	65,536

G.3 Framework Adaptations and Resource Budgets

We adapt LLM4PLC (Fakih et al., 2024), Agents4PLC (Liu et al., 2026), AutoPLC (Yang et al., 2025), and SemaPLC (Tu et al., 2026) to the same public task inputs and plant evaluator. Table 19 summarizes their workflows, checks, and per-task budgets.

Table 19: Workflows, internal checks, and per-task resource limits configured for our PLCWorld adaptations.

Framework	Generation workflow	Internal checks	Configured limits per task
LLM4PLC	Control plan, ST generation, and feedback-based repair	MATIEC compilation; nuXmv on a plan-level SMV model	16 model calls; 8 combined repairs; 700,000 total tokens
Agents4PLC	Retrieval, ranked plans, coding, debugging, and repair	MATIEC compilation; nuXmv on a model translated from supported ST	24 model calls; 3 plans; 2 repairs per plan; 900,000 total tokens
AutoPLC	Planning, case/API retrieval, code generation, and repair	Public submission-profile checks; CODESYS compilation	20 model calls; 3 compiler-feedback repairs; 800,000 total tokens
SemaPLC	Tool-assisted generation, checking, and revision	Native project checks with candidate-specific verification records	60 model calls; 9 candidates; 12 native checks; 1,800 s

Implementation adaptations. LLM4PLC targets the supported ST profile instead of SCL and omits interactive human feedback. Agents4PLC is reconstructed from the paper because the release we used lacks complete orchestration and prompts. AutoPLC replaces hosted retrieval with a deterministic local backend and omits LearnAgent, which requires ground-truth programs. SemaPLC connects its pinned public Bridge/Core implementation and native tools to an isolated WSL worker through a local adapter.

Retrieval and verification details. Agents4PLC uses `text-embedding-ada-002` to retrieve four entries from a fixed local corpus of public language material and generic examples. AutoPLC retains 66 released cases without modification after compatibility filtering. BM25 retrieves five candidates, a model reranker selects three, and API recommendation supplies at most fifteen entries per prompt batch. PLCWorld tasks, reference programs, and outcomes are not used to construct or filter this pool.

Agents4PLC derives verification properties from public requirements without access to the selected plan or code. Invalid properties are discarded without regeneration based on code behavior; candidates with no valid properties remain unverified. Passing these checks does not establish correctness under plant execution.

Final submission and stopping rules. Full-benchmark evaluations use one run per task, with GPT-5.5 repetition 2 as the matched direct baseline. The retained final candidate is evaluated despite

failed internal checks, timeouts, or budget exhaustion; cases without a usable submission count as *Task Success* failures. AutoPLC retains the last candidate, and SemaPLC the latest complete candidate. Documented SemaPLC infrastructure retries replace only affected failed runs. Plant evaluation follows generation and internal verification, and its outcomes are not used for candidate selection or repair.

G.4 Acceptance Checks and Metric Accounting

All metrics use pooled case counts and the error-handling rules in Section 3.3. *Execution Gap* (EG) is conditioned on the common upstream check for the supported ST profile and required I/O interface, applied to the final submitted ST program. We report EG with its eligible-case denominator.

Framework-internal results are reported separately and apply only to the checked program or model. A source-level pass applies to the final submission only when its source matches the checked version. Missing properties, unsupported translations, timeouts, and tool errors do not count as verification passes.

G.5 Statistical Analysis and Efficiency Measurement

Task-cluster bootstrap. We compute 95% SR intervals using 10,000 stratified task-cluster bootstrap replicates (Field & Welsh, 2007) (NumPy default generator; seed `20260920`). Tasks are sampled with replacement within the six suite–difficulty groups, preserving group sizes and all conditions and generation attempts for each sampled task. SR is recomputed from pooled case counts, with intervals given by the 2.5th and 97.5th percentiles. Paired SR differences use identical task resamples. Intervals reflect task resampling conditional on the generated programs and are not adjusted for multiple comparisons.

Efficiency subset. We measure efficiency on 18 shared tasks, sampling three without replacement from sorted task IDs in each of the six suite–difficulty groups. Selection and execution order use seeds `20260921` and `20260922`, respectively. Direct models use the configurations in Table 18; frameworks use GPT-5.5. All measurements use fresh synchronous runs at concurrency one. Batch-job turnaround is not used as per-task latency.

Timing covers model calls, planning, retrieval, checking, and repair, excluding shared initialization and plant evaluation. Direct generation, LLM4PLC, Agents4PLC, and AutoPLC use timers around their generation procedures. SemaPLC uses process-start and final-result file timestamps, which also capture local process and result-writing overhead. These boundary differences limit fine-grained latency comparisons.

Only methods with valid timing observations for all 18 tasks enter the cost comparison. Internal verification failures, timeouts, and budget exhaustion remain included; infrastructure failures are excluded and reported separately.

For each task, we count LLM calls and sum their provider-reported total tokens, excluding embedding calls and tokens. Revisions count repairs or additional ST candidates after initial generation, as recorded by each method. We report medians across the 18 tasks for calls, revisions, tokens, and elapsed time, with the interquartile range (IQR) for time. Table 5 also reports the number of tasks with at least one revision. These measurements describe observed API usage and latency rather than hardware-normalized compute or monetary cost.

Additional operating conditions. The standard evaluation uses 473 registered task–condition pairs. The reported results do not include a separate robustness evaluation under additional held-out operating conditions.

H Additional Experimental Results

This appendix reports additional performance and efficiency results using the protocols described in Appendix G.

H.1 Execution Coverage and Joint Outcomes

Table 20 reports case counts at each evaluation stage, distinguishing failures before plant entry from interrupted observations.

Table 20: Case coverage under the common upstream check. Assigned, check-passed, and plant-entering counts are the SR, EG, and SVR denominators, respectively; these cohorts have no infrastructure exclusions.

Model / Method	Evaluation coverage			Observation status	
	Assigned	Common check passed	Plant entered	Observation complete	Interrupted
(1) Direct ST generation					
GPT-5.5	1419	1399	1399	1399	0
GPT-4o	1419	1062	1021	965	56
Claude Sonnet 5	1419	1383	1383	1382	1
Gemini 3.1 Pro	1419	1316	1316	1316	0
Gemini 3.8 Flash	1419	1187	1176	1173	3
Qwen3.5-Plus	1419	960	946	936	10
Direct (GPT-5.5, matched)	473	462	462	462	0
(2) Generation-and-verification frameworks (GPT-5.5)					
LLM4PLC	473	473	473	473	0
Agents4PLC	473	473	473	473	0
AutoPLC	473	473	473	473	0
SemaPLC	473	473	473	473	0

Table 21 cross-tabulates *Task Success* and observed *Safety Violation* among plant-entering cases. Task completion can coincide with a safety violation, while task failure can occur without an observed violation. For interrupted cases, the absence of an observed violation does not establish safety over the full evaluation endpoint.

Table 21: Joint task and safety outcomes among plant-entering cases. The four outcome counts sum to the number of entered cases; pre-entry failures are excluded.

Model / Method	*Task Success*		Task failure		Plant entered
	No observed violation	Violation	No observed violation	Violation	
(1) Direct ST generation					
GPT-5.5	869	19	295	216	1399
GPT-4o	77	19	598	327	1021
Claude Sonnet 5	679	47	341	316	1383
Gemini 3.1 Pro	840	91	190	195	1316
Gemini 3.8 Flash	806	57	193	120	1176
Qwen3.5-Plus	482	52	207	205	946
Direct (GPT-5.5, matched)	279	10	94	79	462
(2) Generation-and-verification frameworks (GPT-5.5)					
LLM4PLC	265	13	117	78	473
Agents4PLC	241	8	147	77	473
AutoPLC	277	21	117	58	473
SemaPLC	258	6	125	84	473

H.2 Confidence Intervals and Generation Variability

Tables 22 and 23 report pooled SR and paired SR differences with 95% confidence intervals, using the task-cluster bootstrap described in Appendix G.5. These intervals quantify task-resampling uncertainty conditional on the evaluated programs.

Table 22: Pooled SR and task-cluster bootstrap 95% confidence intervals. Direct models use three repetitions (1,419 cases); frameworks use one (473 cases).

Model / Method	Successful / Assigned	SR (%)	95% CI (%)
(1) Direct ST generation			
GPT-5.5	888/1,419	62.58	[55.28, 69.92]
GPT-4o	96/1,419	6.77	[4.38, 9.21]
Claude Sonnet 5	726/1,419	51.16	[45.01, 57.09]
Gemini 3.1 Pro	931/1,419	65.61	[58.72, 72.52]
Gemini 3.8 Flash	863/1,419	60.82	[55.49, 66.24]
Qwen3.5-Plus	534/1,419	37.63	[31.80, 43.26]
(2) Generation-and-verification frameworks (GPT-5.5)			
LLM4PLC	278/473	58.77	[50.74, 67.11]
Agents4PLC	249/473	52.64	[44.66, 61.09]
AutoPLC	298/473	63.00	[54.51, 71.55]
SemaPLC	264/473	55.81	[48.53, 63.25]

Table 23: Paired differences in direct-generation SR, in percentage points (pp), using the same task clusters in both models. Intervals are pointwise and unadjusted.

Comparison	Difference (pp)	95% CI (pp)
(1) Direct ST generation		
GPT-5.5 − GPT-4o	55.81	[48.61, 63.19]
GPT-5.5 − Claude Sonnet 5	11.42	[4.71, 18.67]
GPT-5.5 − Gemini 3.1 Pro	−3.03	[−9.87, 4.29]
GPT-5.5 − Gemini 3.8 Flash	1.76	[−5.27, 8.98]
GPT-5.5 − Qwen3.5-Plus	24.95	[17.59, 32.78]
GPT-4o − Claude Sonnet 5	−44.40	[−50.11, −38.66]
GPT-4o − Gemini 3.1 Pro	−58.84	[−65.85, −52.11]
GPT-4o − Gemini 3.8 Flash	−54.05	[−59.63, −48.57]
GPT-4o − Qwen3.5-Plus	−30.87	[−35.97, −25.76]
Claude Sonnet 5 − Gemini 3.1 Pro	−14.45	[−21.13, −8.22]
Claude Sonnet 5 − Gemini 3.8 Flash	−9.65	[−15.80, −3.89]
Claude Sonnet 5 − Qwen3.5-Plus	13.53	[8.48, 18.61]
Gemini 3.1 Pro − Gemini 3.8 Flash	4.79	[−1.76, 10.99]
Gemini 3.1 Pro − Qwen3.5-Plus	27.98	[21.30, 35.13]
Gemini 3.8 Flash − Qwen3.5-Plus	23.19	[16.88, 29.73]

Table 24 compares each framework's SR with both the matched direct run and the three-run direct baseline using paired task-cluster resampling. All intervals against the matched run include zero; against the three-run baseline, the intervals for Agents4PLC and SemaPLC are below zero.

Table 25 reports SR for three independent generation repetitions, all included in the pooled estimate. With only three repetitions, these results provide a descriptive view of generation variability.

Table 24: Framework-minus-direct GPT-5.5 differences in *Task Success* rate (pp), with pointwise 95% percentile CIs from 10,000 paired task-cluster bootstrap draws stratified by suite and difficulty. Draws retain all task conditions; the three-run baseline averages all direct repetitions. Intervals exclude framework rerun variability.

Framework	Matched direct run difference [95% CI]	Three-run direct mean difference [95% CI]
LLM4PLC	-2.33 [-10.38, 5.91]	-3.81 [-11.06, 3.04]
Agents4PLC	-8.46 [-17.87, 1.06]	-9.94 [-17.93, -2.31]
AutoPLC	1.90 [-6.29, 10.35]	0.42 [-7.47, 8.37]
SemaPLC	-5.29 [-13.38, 2.60]	-6.77 [-13.72, -0.21]

Table 25: SR (%) by generation repetition (100 programs and 473 evaluation cases each). Pooled SR includes all three repetitions.

Model	Repetition			Pooled
	1	2	3	
(1) Direct ST generation				
GPT-5.5	63.21	61.10	63.42	62.58
GPT-4o	5.92	5.71	8.67	6.77
Claude Sonnet 5	49.05	53.70	50.74	51.16
Gemini 3.1 Pro	68.29	63.64	64.90	65.61
Gemini 3.8 Flash	63.21	58.56	60.68	60.82
Qwen3.5-Plus	39.53	35.31	38.05	37.63

H.3 Framework Evaluation Outcomes

Table 26 reports overall SR, SVR, and EG for the adapted frameworks and the matched direct-generation baseline, all using GPT-5.5. Results reflect each method under its configured resource budget. EG is conditioned on the common upstream check.

Table 26: Comparison with GPT-5.5 as the shared backbone. Each row uses one program per task and 473 assigned cases. EG cases give the numerator/denominator under the common upstream check.

Method	SR (%)	SVR (%)	EG (%)	EG cases
(1) Direct ST generation				
Direct (GPT-5.5, matched)	61.10	19.26	39.61	183/462
(2) Generation-and-verification frameworks (GPT-5.5)				
LLM4PLC	58.77	19.24	43.97	208/473
Agents4PLC	52.64	17.97	49.05	232/473
AutoPLC	63.00	16.70	41.44	196/473
SemaPLC	55.81	19.03	45.45	215/473

Framework-internal checks and downstream outcomes. Table 27 links recorded workflow or check status to plant outcomes for the frozen final submissions. Program hashes match the evaluated ST submissions; neither generation nor plant evaluation was rerun. The rows partition each method's 100 tasks, but status labels have method-specific meanings.

The checks do not share a common verification target. LLM4PLC's formal acceptance concerns an LLM-written plan-level SMV model, not a model derived from the final ST; its separate MATIEC

Table 27: Framework status and plant-level outcomes. SR is successful/assigned cases; SVR is cases with an observed safety violation/plant-entering cases. All 473 cases per method entered the plant. Status groups are descriptive and differ in task composition.

Method	Recorded status	Tasks	SR	SVR
LLM4PLC	Plan formal accepted	1	3/3	0/3
	Formal budget	12	42/43	0/43
	Syntax budget	1	4/4	0/4
	Timeout	63	207/343	70/343
	Tool error	23	22/80	21/80
Agents4PLC	Public properties verified	11	37/40	0/40
	Plans exhausted: violated	14	49/53	0/53
	Plans exhausted: vacuous	6	16/26	0/26
	Timeout	69	147/354	85/354
AutoPLC	CODESYS compiled	94	289/452	62/452
	Verification budget exhausted	6	9/21	17/21
SemaPLC	Native checks passed	89	241/414	81/414
	Native compile failed	4	3/26	0/26
	Native compile unchecked	7	20/33	9/33

check accepts 99/100 final programs. Agents4PLC checks public properties over its supported code-derived ST subset. AutoPLC records CODESYS (CODESYS Group, n.d.) compilation, whereas SemaPLC records its native compile and runtime checks. A status without a recorded pass does not imply that no intermediate check was attempted. Because status groups have different task mixes, their outcome differences do not identify the effect of successful verification.

Among compiler-accepted final programs, 208/469 LLM4PLC cases and 179/452 AutoPLC cases have task failure, an observed safety violation, or both. These adverse counts are not the complements of SR because successful cases can also violate safety constraints.

The pinned MATIEC adapter rejects 31 of the 100 reference programs: 19 involve conversion notation and 12 involve mixed-type numeric expressions. Diagnostic copies using explicit conversion functions and operand types compile in all 31 cases. The rewrites change only conversion form and operand typing; we report compilation compatibility, not execution equivalence. We therefore use MATIEC for cross-runtime comparison within the mutually accepted subset, rather than as the common upstream check. The common submission-profile check and its *Execution Gap* therefore remain the cross-method comparison in Table 4; the status breakdown gives method-specific evidence about final submissions.

H.4 Generation Efficiency

Table 28 reports generation efficiency for the six direct-generation models and four GPT-5.5-based frameworks on the shared 18-task subset. Measurement procedures and method-specific timing boundaries are described in Appendix G.5.

These efficiency measurements use fresh runs on the 18-task subset, separate from the 100-task benchmark evaluations. Six of the 18 Gemini 3.8 Flash runs reached the output-token limit configured for the efficiency evaluation. All 18 LLM4PLC efficiency runs ended in a timeout, tool error, or budget exhaustion. The reported costs therefore include incomplete verification workflows. These stopping outcomes should not be interpreted as termination rates for the full-benchmark evaluations.

Table 28: Generation efficiency on the matched 18-task subset. Entries are medians; tokens are in thousands (k) and time in seconds. Time IQR gives the first and third quartiles, computed by inclusive linear interpolation.

Method	Calls	Revisions	Tokens (k)	Time (s)	Time IQR (s)
(1) Direct ST generation					
GPT-5.5	1	0	11.51	99.29	[69.73, 124.38]
GPT-4o	1	0	4.45	5.90	[4.71, 9.42]
Claude Sonnet 5	1	0	17.00	122.32	[49.91, 278.41]
Gemini 3.1 Pro	1	0	19.17	118.09	[49.06, 189.04]
Gemini 3.8 Flash	1	0	25.80	65.54	[20.06, 90.19]
Qwen3.5-Plus	1	0	19.80	250.76	[153.32, 324.74]
(2) Generation-and-verification frameworks (GPT-5.5)					
LLM4PLC	4	0	48.28	321.13	[254.40, 380.97]
Agents4PLC	5	1	73.11	452.56	[369.00, 760.94]
AutoPLC	4	0	30.96	134.34	[90.71, 150.86]
SemaPLC	18	1	523.88	230.90	[157.75, 275.87]

I Execution Gap and Failure Analysis

I.1 Common Upstream Check and Execution Gap Counts

The common upstream check used for Table 4 is PLCWorld's local submission-profile check, recorded as `local_submission_profile`. It parses the submitted ST and checks the required program structure, declared I/O types, read-only inputs, and supported constructs. This check precedes runtime initialization and plant execution; it is neither external native compilation nor a framework-specific verification result.

The check result is linked to the same final ST program subsequently evaluated by PLCWorld using a program hash. When a task-level submission passes, all registered execution conditions assigned to that same program enter the case-level acceptance set in Eq. (3). Thus, the *Execution Gap* denominator counts accepted task–condition cases, not distinct programs. Subsequent initialization or execution errors attributable to the submitted program count as task failures among accepted cases. Table 29 gives the numerator and denominator of the values in the main table; accepted counts should be considered when comparing conditional gaps.

Table 29: *Execution Gap* under the common submission-profile check. Accepted and adverse counts refer to metric-eligible task–condition cases. Adverse cases have *Task Success* failure, an observed *Safety Violation*, or both.

Model / method	Assigned	Accepted	Adverse among accepted	Execution Gap (%)
(1) Direct ST generation				
GPT-5.5	1,419	1,399	530	37.88
GPT-4o	1,419	1,062	985	92.75
Claude Sonnet 5	1,419	1,383	704	50.90
Gemini 3.1 Pro	1,419	1,316	476	36.17
Gemini 3.8 Flash	1,419	1,187	381	32.10
Qwen3.5-Plus	1,419	960	478	49.79
(2) Generation-and-verification frameworks (GPT-5.5)				
Direct (GPT-5.5, matched)	473	462	183	39.61
LLM4PLC	473	473	208	43.97
Agents4PLC	473	473	232	49.05
AutoPLC	473	473	196	41.44
SemaPLC	473	473	215	45.45

Table 30: Mapping of ten rule-level classes to eight evaluator-recorded failure signatures. `support_prohibited` is mapped according to its recorded reason.

Signature	Detailed rule class	Grouping criterion
Completion protocol		
Completion timing	`absent_late`	Missing or late completion
Completion validity	`premature_false`	Invalid early declaration
Completion retention	`persistence`	Lost required state or signal
Safety constraints		
Actuation constraints	`command_conflict`; `support_prohibited` for prohibited-operation reasons	Command or interlock violation
Spatial constraints	`collision_clearance`	Collision, clearance, or travel
Resource constraints	`occupancy_capacity`; `support_prohibited` for support or material-integrity reasons	Capacity, holding, or support
Required behavior		
Goal fulfillment	`request_ack`, `physical_goal`	Unmet request or physical goal
Process conformance	`sequence_route`	Order, route, identity, or quantity

Framework-internal checks are not pooled with this common check: they may inspect different artifacts or apply different acceptance criteria. The matched direct row uses a single generation per task and is shown for framework comparison; its cases are not added again to the pooled direct-generation results.

I.2 Rule-to-Signature Mapping

Failure signatures describe evaluator-recorded manifestations, not source-code root causes. Signature analysis is restricted to cases that passed the common check, entered plant execution, and either failed the task or incurred an observed safety violation. A case may activate several signatures. Cases that never enter the plant remain in the *Execution Gap* and outcome analyses where applicable, but receive no plant-level signature.

The fixed rule mapping first assigns each failed evaluator rule to one of ten detailed classes and then to one of the eight signatures reported in Section 5.3.2. Table 30 shows this correspondence. The `support_prohibited` class is split by its recorded reason: support and material-integrity reasons map to Resource constraints; other prohibited-operation reasons map to Actuation constraints. This split uses the same fixed reason list for every model and method.

The mapping uses each failed rule's scope, identifier, and recorded reason; the complete rule-identifier and reason inventory is retained in the accompanying analysis artifact. An unseen rule stops the mapping audit rather than entering a catch-all category. An adverse case without failed-rule detail remains in the outcome count but receives no inferred signature; in particular, it is not automatically labeled as missing completion.

I.3 Outcome and Signature Distributions

Tables 20 and 21 report execution coverage and joint outcomes. After passing the common check, 41 GPT-4o, 11 Gemini 3.8 Flash, and 14 Qwen3.5-Plus cases never enter plant execution. They count as failures in *Execution Gap* but have no safety judgment or plant-level signature.

Table 31: Evaluator-observed signature incidence among adverse plant-entered cases (%). CT: completion timing; CV: completion validity; CR: completion retention; GF: goal fulfillment; PC: process conformance; AC: actuation constraints; SC: spatial constraints; RC: resource constraints. ND is the percentage of adverse cases without failed-rule detail. The eight signatures are multi-label; their percentages need not sum to 100%.

Model / method	Adverse n	CT	CV	CR	GF	PC	AC	SC	RC	ND
(1) Direct ST generation										
GPT-5.5	530	81.1	4.2	42.5	4.3	1.3	29.2	8.3	23.4	12.6
GPT-4o	944	74.4	7.6	26.6	4.1	1.7	16.4	8.4	10.9	18.6
Claude Sonnet 5	704	79.3	8.0	40.9	1.6	2.4	27.7	12.1	29.8	13.6
Gemini 3.1 Pro	476	61.3	15.5	41.2	5.0	6.7	32.1	7.1	26.1	17.6
Gemini 3.8 Flash	370	64.9	4.6	39.2	2.7	5.9	26.8	5.7	10.3	23.8
Qwen3.5-Plus	464	63.8	23.7	28.7	6.7	4.7	18.1	13.6	19.4	20.9
(2) Generation-and-verification frameworks (GPT-5.5)										
Direct (GPT-5.5, matched)	183	78.1	8.2	41.0	1.6	2.7	32.2	11.5	29.5	11.5
LLM4PLC	208	77.9	2.4	38.9	1.4	8.7	25.0	5.8	14.9	13.9
Agents4PLC	232	83.6	5.6	38.8	1.3	8.6	25.4	6.0	11.6	11.2
AutoPLC	196	69.4	7.1	42.3	6.1	1.5	28.6	7.7	11.2	13.3
SemaPLC	215	74.0	7.9	47.4	5.6	7.0	30.2	12.1	20.9	16.3

Table 31 reports signatures by model and method among adverse plant-entered cases passing the common check. Each case counts at most once per signature, but labels overlap. The matched direct baseline is counted only once in the pooled 4,339 cases. Of these, 724 lack failed-rule detail and remain in the denominator without assigned signatures. These distributions describe recorded manifestations, not source-code root causes.

I.4 Representative Evaluation Outcomes

Table 32 illustrates distinct outcomes and multi-label mapping. The examples were selected for explanatory diversity rather than frequency.

Table 32: Representative case-level evaluator outcomes. The final column gives observed rule outcomes, not inferred source-code root causes. Here, r01 denotes the first generation run for each model or method.

Model / run	Task / condition	Task Success	Safety Violation	Recorded manifestation
GPT-5.5, r01	MOT-E11 / 0	No	No	Completion is asserted before the required post-transition plant condition; `premature_completion` is recorded.
GPT-5.5, r01	MOT-H07 / 0	Yes	Yes	The task reaches valid completion, but the execution also records `head_collision`.
LLM4PLC, r01	MAT-H06 / 0	No	Yes	`buffer_overflow`, `wrong_piece_length`, `completion_not_held`, and `no_completion` activate several signatures in one case.

The stored case records identify the frozen program hash, registered condition, evaluator version, failed rules, and execution-trace hash. These observations establish the reported outcomes; identifying a source-code root cause would additionally require inspection of the submitted ST and scan-level trace.