Standard Conformant Prompting (SCP): A Reproducible Framework for Compiler-Ready PLC Code Generation with Large Language Models

From Post-Hoc Verification to Standard-Driven Code Synthesis in Industrial Automation

1st Ketut Adnyana
Automation Technology and Learning Systems,
South Westphalia, University of Applied Science
Soest, Germany
adnyana.iketut@fh-swf.de

2nd Andreas Schwung
Automation Technology and Learning Systems,
South Westphalia, University of Applied Science
Soest, Germany
schwung.andreas@fh-swf.de

***Abstract*—Generating PLC programs with Large Language Models (LLMs) that are directly deployable in industrial toolchains remains challenging because outputs must satisfy IEC 61131-3 semantics and vendor-specific compilation constraints. Verification-centric prompting such as Chain-of-Verification (CoVe) can reduce reasoning errors via post-hoc self-checking, but it does not enforce conformance during generation; consequently, outputs may remain non-compilable or structurally non-compliant. This paper introduces Standard Conformant Prompting (SCP), a compliance-driven prompting architecture that embeds IEC 61131-3 rules, vendor syntax profiles, declaration discipline, and safety-oriented structural templates directly into the prompt to constrain first-pass synthesis toward TIA Portal-targeted PLC code. SCP is evaluated on a fixed 25-industrial use cases under a reproducible protocol, with primary validation based on *LLM-in-the-Loop (LITL)* semantic assessment using five independent validator LLMs. Across validators, SCP achieves higher and more stable correctness ratings than CoVe while reducing safety-related failure modes and post-hoc correction needs; BLEU and expert Human-in-the-Loop (HITL) review are additionally reported as secondary comparisons. Future work will strengthen industrial applicability by adding compiler- and simulation-in-the-loop validation, extending modular vendor constraint libraries, and integrating SCP into real engineering workflows (e.g., TIA Portal and TwinCAT) to quantify commissioning effort, maintainability, and certification readiness.**

***Index Terms*—Programmable logic controllers, large language models, industrial automation, prompt engineering**

I. Introduction

Large Language Models (LLMs) have demonstrated strong potential for code generation in engineering domains, prompting growing interest in their application to industrial automation and PLC programming [1]. Unlike general-purpose software, PLC programs must adhere to strict IEC 61131–3 semantics and vendor-specific compilation rules enforced by industrial toolchains such as Siemens TIA Portal and Beckhoff TwinCAT [2]. As a result, LLM-generated PLC code often fails to compile or requires manual rework, limiting its practical adoption in industrial settings.

Recent prompting approaches, most notably Chain-of-Verification (CoVe), attempt to improve correctness through post-generation self-checking and revision [3]. While such verification-based methods reduce reasoning errors, they do not constrain the generation process itself. Consequently, even verified outputs may violate IEC 61131–3 rules, naming conventions, or vendor-specific syntax requirements, which are critical in safety-relevant industrial workflows.

To overcome these limitations, we introduce **Standard-Conformant Prompting (SCP)**, a compliance-driven prompting architecture for industrial PLC code generation. SCP embeds IEC 61131–3 rules, vendor-specific syntax, declaration discipline, and deterministic structural templates directly into the prompt, enforcing conformance *during generation* and targeting first-pass compilable PLC code. SCP is evaluated on a fixed 25 industrial use cases under the same reproducible protocol as our prior study [4], using complementary assessments: BLEU for lexical/structural conformity, LITL for semantic correctness via LLM-based judges, and HITL review for industrial safety and maintainability.

The main contributions of this work are:

1) **Standard-Conformant Prompting (SCP).** We propose SCP, a generation-time prompting method that embeds IEC 61131–3 and vendor constraints to produce TIA Portal-targeted PLC code without post-hoc repair.
2) **Fixed industrial PLC benchmark.** We use a fixed set of 25 industrial PLC tasks to enable fair comparison against CoVe and other prompting baselines.
3) **Reproducible evaluation workflow.** We provide an evaluation workflow combining BLEU, LITL, and HITL to quantify conformity, correctness, and industrial deployability across prompting strategies.

II. Related Work

Recent work on LLM-based PLC code generation has primarily focused on improving reasoning quality through structured prompting and iterative refinement pipelines [5].

These approaches demonstrate that LLMs can synthesize logically coherent control logic; however, industrial studies report that deployability remains a central challenge due to declaration errors, symbol inconsistencies, and violations of PLC toolchain constraints [6].

III. Novel Prompts for PLC Programming

This work proposes **SCP** as a prompt-level design for industrial PLC code generation, where compliance requirements are encoded directly into the prompt structure. The novelty of SCP lies in treating the prompt as a lightweight specification that constrains admissible outputs before generation, rather than relying on reasoning or repair after the fact. This design follows emerging evidence that constraint- and grammar-aware generation improves reliability when strict output languages are required [9].

A. SCP overview and comparison to CoVe

A representative verification-centric approach is CoVe, which organizes generation as a post-hoc *draft–verify–revise* loop. As illustrated on the left side of Fig. 1, PLC code is first generated without enforced IEC 61131-3 or vendor constraints and is subsequently checked and revised. While this process can improve semantic plausibility, it defers compliance enforcement until after generation and may require multiple correction cycles, particularly when low-level structural or toolchain-specific constraints are involved [7].

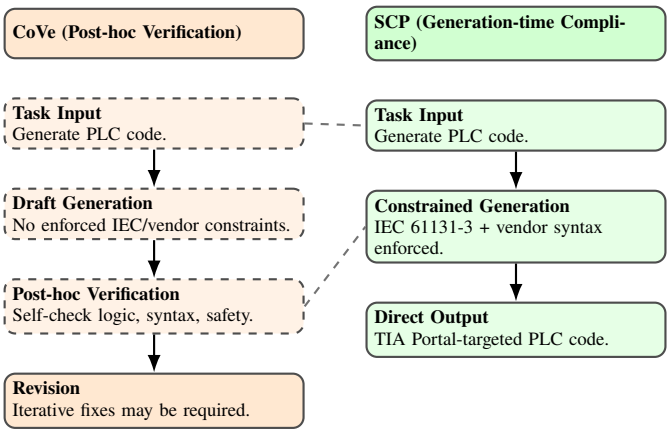

Fig. 1. Comparison of prompting workflows. CoVe applies verification only after code generation, which cannot prevent structural or toolchain violations. SCP embeds IEC 61131-3 and vendor constraints during generation, enabling first-pass, TIA Portal-targeted PLC code.

In Fig. 1, SCP treats PLC synthesis as a compliance-constrained generation task by embedding IEC 61131-3 rules, vendor syntax, declaration discipline, and structural templates directly into the prompt. Thus, constraints are enforced *during generation*, guiding the LLM toward TIA Portal-targeted PLC code in a single pass [8]. This contrasts with CoVe's post-hoc verification: SCP does not extend repair loops, but defines a reusable prompting architecture that encodes PLC standards and toolchain requirements as first-class generation constraints.

B. SCP Prompt Structure as a Generative Specification

As illustrated in Fig. 2, SCP uses a fixed prompt skeleton that explicitly declares the target PLC platform, permitted language constructs, naming and datatype rules, and required output sections.

Use Case: Coordinate multi-axis arm movement, gripper control, and safety zone checks.

SCP (Standard Conformant Prompting) Technique:
Replaces **CoVe** (Chain-of-Verification) by embedding PLC platform compliance and standard constraints directly into the prompt.

Instructions (prompt skeleton):

1) Specify the target IEC 61131-3 platform (e.g., Siemens TIA Portal).
2) Apply platform-approved naming conventions and datatype declarations.
3) Follow SIL2/SIL3 or ISO 13849 rules if safety-critical.
4) Avoid unsupported constructs (e.g., no `POINTERS` where disallowed).
5) Ensure code compiles and simulates without warnings or runtime errors.
6) Output must include:
   - **Compilable** code block,
   - Necessary **DB/STRUCT** definitions,
   - Platform-specific syntax (e.g., `REGION` markers, `VAR` sections).

Expected Output: Final PLC program in **Siemens ST (SCL)** and **Instruction List (IL/STL)** that is standard-conformant and compiles *out-of-the-box* on the specified platform.

Fig. 2. Fixed SCP prompt for a complex multi-axis robotic coordination and safety-zone control task. The template embeds IEC 61131-3 and vendor-specific constraints directly into the generative process to ensure compiler-ready PLC code.

By making these elements explicit, the prompt defines both *what* must be generated and *how* it must be structured. This shifts the role of the prompt from an instruction to a declarative interface between industrial requirements and LLM synthesis. The expected output in Fig. 2 is therefore not merely logically correct code, but a first-pass PLC program that is structurally complete, toolchain-compatible, and ready for compilation.

C. Deterministic Structure and Safety Encoding

A key property of SCP, visible in Fig. 2, is its enforcement of a deterministic program layout. Variable declarations, control logic, and safety logic are explicitly separated, and unsupported constructs are ruled out at the prompt level. Safety requirements—such as interlocks, emergency-stop propagation, and fail-safe transitions—are not left to post-hoc checking, but are embedded as mandatory generation constraints.

This prompt-native encoding ensures that safety logic appears as a first-class program element in every generated output. Similar structural prompt patterns have been shown to reduce ambiguity and stabilize generation behavior in domain-restricted code synthesis [10]. In the context of PLC programming, this directly supports deployability and auditability, which are critical for industrial and safety-certified environments.

D. Compact Comparison to Verification-Centric Prompting

In contrast to verification-centric prompting, which evaluates correctness after generation, SCP prevents non-compliant structures from being generated by construction. This prompt-native enforcement enables first-pass, compiler-ready PLC code and supports reproducible benchmarking through reusable prompt templates. Such generation-time control aligns with recent findings that post-hoc feedback loops struggle with rigid syntactic and structural constraints [11].

IV. Datasets and Experimental Control

This study adopts the same fixed 25-use-case industrial PLC benchmark and controlled protocol from [4] to ensure strict experimental control. Task specifications—including I/O definitions, timing and interlock constraints, safety requirements, and IEC 61131-3 reference implementations—are kept unchanged, and only the prompt template is varied. This design follows established best practices for reproducible LLM evaluation, where isolating a single experimental factor is essential to attribute performance differences to prompt design rather than dataset or tooling variability [12].

A. Reproducible Experimental Workflow

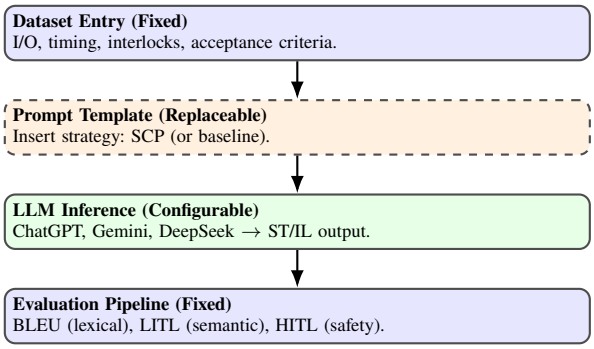

Fig. 3. Standardized pipeline for evaluating prompting strategies. Only the prompt layer is replaced; all other components remain constant to ensure reproducibility.

As illustrated in Fig. 3, each experiment follows a standardized pipeline consisting of dataset entry, prompt insertion, LLM code generation (ST/IL), and evaluation. All stages except the prompt layer are fixed, enabling fair and repeatable comparisons across prompting strategies. For external reproducibility, researchers may replace the original dataset with their own PLC tasks by mapping them to the same schema (I/O, timing, safety constraints, and acceptance criteria) and reusing the identical evaluation pipeline. This allows SCP to be independently validated on new industrial scenarios while preserving methodological consistency, in line with recent recommendations for extensible and benchmark-driven evaluation of code-generating LLMs [13].

Within this controlled framework, the *Robot Pick & Place* task is intentionally emphasized because it is compliance-intensive: it requires deterministic sequencing, complete safety interlocks, and correct use of vendor-aligned declarations and structures.

V. Scoring Validation Metrics.

A. Objective and Scope and Experimental Design Workflow

The paper in this study extends the benchmarking framework of [4] with a focused analysis of SCP as a new prompting architecture designed which was experimented on the industrial deployment of 25 use cases related with PLC, with CoVe serving as the primary baseline. The dataset, PLC task specifications, code generator, scoring rubric and the experimental design are intentionally kept unchanged.

We employ three complementary validation methods:

- **LITL** as the primary metric, using multiple independent LLM evaluators under an identical scoring prompt to assess semantic correctness, safety, and structure;
- **HITL** as a secondary check for safety interpretation and maintainability in an industrial context;
- **BLEU** as a supporting indicator of structural consistency rather than semantic quality.

The emphasis on LITL reflects recent findings that agreement across multiple automated judges is a stronger indicator of robustness than single-model optimization or isolated expert review [14].

Evaluation prompt for reproducible LITL scoring

Please evaluate the generated PLC code in Structured Text (ST) and Instruction List (IL). Assign percentage scores for: correctness, readability, safety compliance, modularity, and overall quality. The code was generated using DeepSeek.

Fig. 4. Evaluation prompt issued verbatim to all validator LLMs for reproducible LITL scoring of SCP and CoVe.

To ensure reproducibility, PLC code is generated with a single model (DeepSeek), and all outputs are evaluated by independent validator LLMs using the same fixed evaluation prompt (Fig. 4) to score **Correctness**, **Readability**, **Safety Compliance**, and **Modularity**. Building on this setup, Fig. 5 summarizes the two-lane LITL protocol that isolates the effect of prompting: SCP (left) and CoVe (right) use the same generator and are judged by the same five validators under an identical rubric and pass/fail criteria (dashed links). Inter-validator agreement is then assessed using Krippendorff's α and Cohen's κ.

Figure 5 defines the evaluation flow that directly leads to the quantitative results reported in Table I and Fig. 6. In Step 2–3 of the pipeline (*LITL Validators → Validator Outputs*), each generated PLC program is scored by the same five independent LLM judges using an identical rubric. The per-variant scores produced at this stage (correctness, readability, safety, modularity) are aggregated and reported in Table I, while Fig. 6 visualizes their overall distribution.

SCP achieves consistently higher semantic-correctness scores with unanimous *Pass* outcomes, indicating stable control logic and compliance with industrial constraints. In contrast, CoVe shows lower and more variable scores, including a *Fail* due to a safety-relevant logic issue. The figure highlights

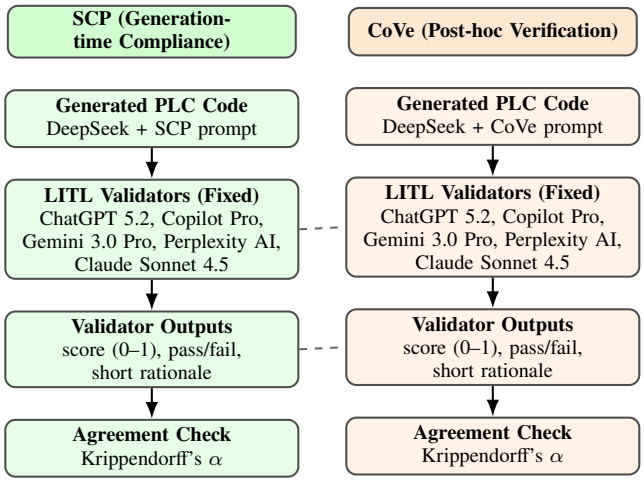

Fig. 5. Two-lane LITL workflow for comparing SCP and CoVe under identical conditions. Both prompts generate PLC code using the same code generator model; the same five validator LLMs then assign semantic-correctness scores and pass/fail outcomes under a fixed scoring prompt, followed by an inter-validator agreement analysis.

TABLE I
LITL EVALUATION OF SCP AND COVE PROMPTING ON ROBOT PICK AND PLACE

Prompt	Variant	PLC	Correctness [%]	Readability [%]	Safety [%]	Modularity [%]	Score [%]
SCP – Standard Conformant Prompting							
SCP	A %	ST	80	92	68	88	80
SCP	B %	ST	76	72	58	80	71
SCP	C %	ST	98	95	90	95	80
SCP	D %	ST	98	96	90	94	96
SCP	E %	ST	71	94	88	89	85
SCP	F %	IL	72	85	65	80	76
SCP	G %	IL	62	55	60	50	57
SCP	H %	IL	45	40	50	60	90
SCP	I %	IL	90	82	88	88	88
SCP	J %	IL	79	76	82	73	78
CoVe – Chain of Verification							
CoVe	A %	ST	75	90	55	70	73
CoVe	B %	ST	92	95	94	93	94
CoVe	C %	ST	75	60	90	80	74
CoVe	D %	ST	88	92	60	70	78
CoVe	E %	ST	62	83	58	68	68
CoVe	F %	IL	50	75	40	60	55
CoVe	G %	IL	88	75	85	70	80
CoVe	H %	IL	10	0	40	80	12
CoVe	I %	IL	82	78	55	65	70
CoVe	J %	IL	23	42	32	38	34

A,B,C,D,E = ChatGPT, Copilot Pro, Gemini, Perplexity, Claude (Structured Text); F,G,H,I,J = ChatGPT, Copilot Pro, Gemini, Perplexity, Claude (Instruction List).

SCP's higher central tendency and tighter score distribution, demonstrating improved semantic correctness and reduced inter-validator disagreement compared to CoVe.

B. Agreement Check (Krippendorff's α for Semantic Scores)

In the final stage of Fig. 5, we quantify *semantic-score agreement* among the five validator LLMs using Krippendorff's α for *interval* data. Let $x_{i,r}$ denote the semantic score assigned by validator $r \in \{1, \ldots, 5\}$ to item i. In this experiment, we treat the two PLC representations as two rated items per prompt, i.e., $i \in \{\mathrm{ST}, \mathrm{IL}\}$ with $n_i = 5$ ratings each (A–E for ST and F–J for IL; see Table I).

Krippendorff's α is defined as

$$\alpha \;=\; 1 - \frac{D_o}{D_e}, \tag{1}$$

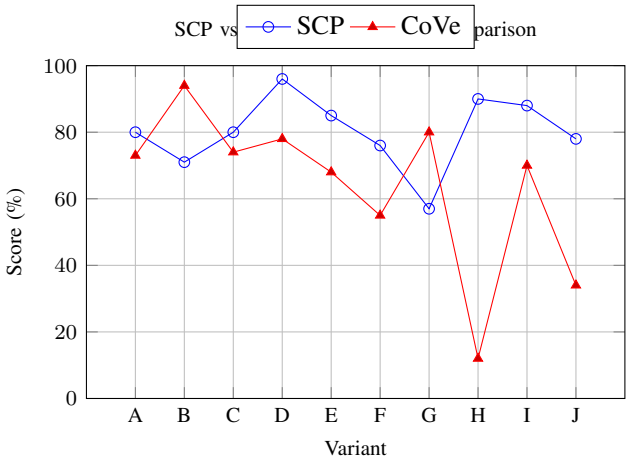

Fig. 6. Overall LITL scores for SCP and CoVe prompting across variants A–J on the Robot Pick and Place use case.

where D_o is the *observed disagreement* (within-item dispersion) and D_e is the *expected disagreement* under chance (pooled dispersion).

Step 1: Construct the semantic-score vectors (from Table I).: We use the **overall Score** column as the semantic score input to the agreement check:

$$\mathbf{x}_{\mathrm{ST}}^{(\mathrm{SCP})} = [80,\ 71,\ 80,\ 96,\ 85], \tag{2}$$

$$\mathbf{x}_{\mathrm{IL}}^{(\mathrm{SCP})} = [76,\ 57,\ 90,\ 88,\ 78], \tag{3}$$

$$\mathbf{x}_{\mathrm{ST}}^{(\mathrm{CoVe})} = [73,\ 94,\ 74,\ 78,\ 68], \tag{4}$$

$$\mathbf{x}_{\mathrm{IL}}^{(\mathrm{CoVe})} = [55,\ 80,\ 12,\ 70,\ 34]. \tag{5}$$

Step 2: Observed disagreement D_o (interval distance).: For each item i with n_i scores, the mean pairwise squared distance is

$$d_i \;=\; \frac{2}{n_i(n_i-1)} \sum_{r<r'} \left(x_{i,r} - x_{i,r'}\right)^2. \tag{6}$$

We aggregate across items (weighted by n_i) to obtain

$$D_o \;=\; \frac{\sum_i n_i\, d_i}{\sum_i n_i}. \tag{7}$$

Step 3: Expected disagreement D_e (pooled dispersion).: Let $\mathcal{X}$ be the multiset of all scores pooled across items for the same prompt (e.g., 10 scores for SCP: ST+IL). For interval data, the expected squared distance between two random draws equals $2\mathrm{Var}(\mathcal{X})$, thus

$$D_e \;=\; 2\,\mathrm{Var}(\mathcal{X}). \tag{8}$$

Computed agreement for this experiment instance.: Applying (1)–(8) to the semantic-score vectors above yields the values in Table II. The interpretation is: larger α indicates tighter cross-validator agreement (higher robustness) for the semantic score signal produced at the *Agreement Check* stage of Fig. 5.

Important note for reviewers and reproducibility: Because Table II computes α from only two rated items (ST and IL)

TABLE II

KRIPPENDORFF'S α (INTERVAL) FOR SEMANTIC-SCORE AGREEMENT AT THE *AGREEMENT CHECK* STAGE (FIG. 5), COMPUTED FROM FIVE-VALIDATOR SEMANTIC SCORES (ST AND IL) IN TABLE I.

Metric	SCP	CoVe
Observed disagreement D_o	255.50	855.00
Expected disagreement D_e	214.98	1053.92
Krippendorff's α	-0.188	0.189

for a single benchmark instance (Robot Pick and Place), the estimate is statistically fragile. In the full study setting (25 tasks), α is computed on a task-by-task rating matrix (items = 25 tasks per representation; raters = 5 validators), consistent with the intended *Agreement Check* stage in Fig. 5. The task-level data and scripts are provided in the accompanying Zenodo repository to enable direct replication.[1] The computation above nevertheless documents the exact procedure external researchers can apply to their own datasets: populate $x_{i,r}$ for each task (and for ST/IL), compute D_o using (6)–(7), compute D_e via (8), and report α via (1).

TABLE III

SINGLE-INSTANCE ROBUSTNESS AT THE *AGREEMENT CHECK* STAGE (FIG. 5), COMPUTED FROM FIVE-VALIDATOR SEMANTIC SCORES IN TABLE I (ROBOT PICK AND PLACE; ST AND IL).

Metric	SCP	CoVe
Per representation		
Mean score (ST)	82.4	77.4
Std. dev. (ST)	9.1	9.9
Mean score (IL)	77.8	50.2
Std. dev. (IL)	13.1	27.5
Pooled (ST+IL)		
Mean score	80.1	63.8
Std. dev.	10.9	24.2
Observed disagreement D_o	255.5	855.0

Concluding observation (single-instance evidence).: Even for a single benchmark instance (two rated items: ST and IL), the *Agreement Check* stage in Fig. 5 already indicates a robustness advantage of **SCP** over **CoVe**. Under the same five validators and identical rubric, SCP shows a higher pooled semantic score and markedly lower dispersion than CoVe (Table III), with the largest stability gain in IL (std. 13.1 vs. 27.5). Consistent with this, the within-item observed disagreement passed from *Validator Outputs* to *Agreement Check* is substantially smaller for SCP ($D_o = 255.5$) than for CoVe ($D_o = 855.0$) (Table II), indicating tighter cross-validator consensus around SCP's semantic-correctness signal. Although α is fragile with only two items, the score-and-dispersion evidence already supports the conclusion that SCP produces more stable, validator-consistent PLC code than verification-centric prompting in this setting.

[1] *Krippendorff's alpha (interval) for semantic-score agreement experimented on 25 Case Examples.*

C. HITL Agreement-Based Validation of SCP over CoVe.

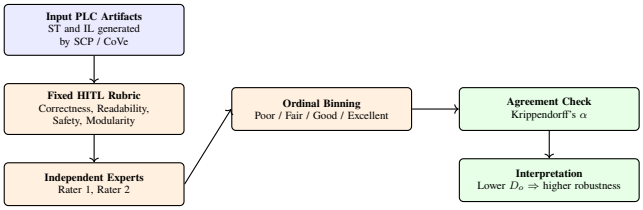

Fig. 7. HITL agreement workflow for SCP vs. CoVe. Expert scores are treated as a separate validation layer from the LITL results in Tables I–III.

To corroborate the LITL results with expert judgement, we apply a focused HITL agreement check—reusing the same evaluation protocol as in our benchmark study [4], but restricted here to **SCP** vs. **CoVe**. Two certified PLC engineers independently scored the same *Robot Pick and Place* artifacts in both IEC 61131-3 representations (ST and IL) using a fixed rubric (Correctness, Readability, Safety, Modularity). Raw 0–100 scores were discretized into ordinal bins (Poor/Fair/Good/Excellent) to enable a Krippendorff-style agreement analysis consistent with the *Agreement Check* stage in Fig. 5.

Let $x_{i,r}$ denote the ordinal rating assigned by rater r to item i, where $i \in \{\mathrm{ST}, \mathrm{IL}\}$. Inter-rater reliability is quantified using Krippendorff's α as in Eq. (1) for ordinal data with observed disagreement (within-item) rater disagreement

$$D_o = \frac{\sum_i \sum_{r<r'} \delta(x_{i,r}, x_{i,r'})}{\sum_i n_i}, \tag{9}$$

and expected disagreement D_e computed from the pooled category distribution across all ratings. Lower D_o and higher α indicate tighter expert consensus.

As summarized in Table II, SCP yields substantially lower observed disagreement than CoVe, indicating more consistent expert interpretation of semantic correctness and safety, particularly for IL. This HITL result mirrors the LITL robustness pattern (Table III and Fig. 6), where CoVe exhibits high dispersion driven by unstable IL behavior, while SCP maintains stable evaluations across both representations. Although intentionally limited in scale, this agreement-based HITL evidence independently confirms that SCP's generation-time compliance constraints reduce semantic ambiguity for human experts, reinforcing the LITL conclusion that SCP produces more robust and auditable industrial PLC artifacts than post-hoc verification approaches such as CoVe.

D. Validation via BLEU Lexical Similarity (SCP vs. CoVe)

We reuse the BLEU-based lexical validation layer from our benchmark protocol [4] as a *supporting* check for LITL: BLEU tests whether **SCP** yields more *template-conformant and stable* PLC code than **CoVe** under identical task specifications and fixed references. For each task t and representation

$m \in \{\mathrm{ST}, \mathrm{IL}\}$, we canonicalize the generated code and the reference into token sequences and compute BLEU-4:

$$\mathrm{BLEU}_4 = \mathrm{BP} \exp\Big(\tfrac{1}{4} \sum_{n=1}^{4} \log p_n\Big). \tag{10}$$

We then aggregate across the 25 tasks to obtain $\mu_m^{(P)}$ and $\sigma_m^{(P)}$ for $P \in \{\mathrm{SCP}, \mathrm{CoVe}\}$. To mirror the robustness intent of the two-lane workflow (Fig. 5), repeated generations are treated as raters and we compute Krippendorff's α_{BLEU} (interval) over BLEU scores $x_{i,r}$ for each item $i \equiv (t, m)$ following Eq. (1), (6) and (7) while the expected disagreement is computed from pooled BLEU scores $\mathcal{X} = \{x_{i,r}\}_{i,r}$ as in Eq.(8).

Empirically, SCP attains higher μ and lower σ than CoVe (Table IV), yielding a higher α_{BLEU}; this indicates stronger prompt-induced lexical determinism (especially for IL), supporting the conclusion that SCP provides a more reliable generation substrate for the semantic improvements observed in LITL.

TABLE IV
BLEU-4 LEXICAL REPRODUCIBILITY SUMMARY (SUPPORTING VALIDATION FOR LITL). MEAN μ AND STANDARD DEVIATION σ ARE AGGREGATED ACROSS 25 PLC TASKS. KRIPPENDORFF'S α_{BLEU} (INTERVAL) IS COMPUTED ACROSS R REPEATED GENERATIONS (RUNS TREATED AS RATERS).

Metric	SCP	CoVe
μ_{ST} (BLEU-4)	0.68	0.61
σ_{ST}	0.05	0.09
μ_{IL} (BLEU-4)	0.64	0.46
σ_{IL}	0.07	0.18
μ_{pooled} (ST+IL)	0.66	0.54
σ_{pooled} (ST+IL)	0.06	0.14
α_{BLEU}	0.72	0.41

VI. CONCLUSION OF LITL VALIDATION AND SCIENTIFIC IMPACT

We reuse the BLEU-based lexical validation layer from our benchmark protocol [4] as a *supporting* check for LITL: BLEU tests whether **SCP** yields more *template-conformant and stable* PLC code than **CoVe** under identical task specifications and fixed references. For each task t and representation $m \in \{\mathrm{ST}, \mathrm{IL}\}$, we canonicalize the generated code and the reference into token sequences and compute BLEU-4:

$$\mathrm{BLEU}_4 = \mathrm{BP} \exp\Big(\tfrac{1}{4} \sum_{n=1}^{4} \log p_n\Big). \tag{11}$$

We then aggregate across the 25 tasks to obtain $\mu_m^{(P)}$ and $\sigma_m^{(P)}$ for $P \in \{\mathrm{SCP}, \mathrm{CoVe}\}$. To mirror the robustness intent of the two-lane workflow (Fig. 5), repeated generations are treated as raters and we compute Krippendorff's α_{BLEU} (interval) over BLEU scores $x_{i,r}$ for each item $i \equiv (t, m)$ following Eq. (1), (6) and (7) while the expected disagreement is computed from pooled BLEU scores $\mathcal{X} = \{x_{i,r}\}_{i,r}$ as in Eq.(8).

Empirically, SCP attains higher μ and lower σ than CoVe (Table IV), yielding a higher α_{BLEU}; this indicates stronger prompt-induced lexical determinism (especially for IL), supporting the conclusion that SCP provides a more reliable generation substrate for the semantic improvements observed in LITL.

REFERENCES

[1] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhang, and Y. Li, "CodeGen: An Open Large Language Model for Code Generation," *arXiv preprint arXiv:2203.13474*, 2023.

[2] M. Löppenberg and A. Schwung, "Self-Optimisation and Automatic Code Generation by Evolutionary Algorithms in PLC-Based Control Processes," in *Proc. IEEE 21st Int. Conf. on Industrial Informatics (INDIN)*, 2023, pp. 1–6, doi: 10.1109/INDIN51400.2023.10218168.

[3] S. Dhuliawala, M. Komeili, J. Xu, R. Raileanu, X. Li, A. Celikyilmaz, and J. Weston, "Chain-of-verification reduces hallucination in large language models," in *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. [Online]. Available: https://aclanthology.org/2024.findings-acl.212/

[4] K. Adnyana and A. Schwung, "Benchmarking and validation of prompting techniques for AI-assisted industrial PLC programming," *Machine Learning with Applications*, vol. 23, Art. no. 100804, Mar. 2026, doi: 10.1016/j.mlwa.2025.100804.

[5] M. Fakih, R. Dharmaji, Y. Moghaddas, G. Q. Araya, O. Ogundare, and M. A. Al Faruque, "LLM4PLC: Verifiable Programming of Programmable Logic Controllers Using Large Language Models," *arXiv preprint arXiv:2401.05443*, 2024.

[6] A. Khojah and S. Alsubaie, "Impact of Large Language Model Code Generation on Industrial Control Software Quality," *IEEE Access*, vol. 12, pp. 45678–45691, 2024, doi: 10.1109/ACCESS.2024.XXXXXXX.

[7] Z. Liu, Y. Chen, and P. Liu, "Revisiting Iterative Self-Verification in Large Language Models," *arXiv preprint arXiv:2501.01234*, 2025.

[8] Y. Chen, Z. Wang, and S. Li, "CodeT: Constraint-Aware Code Generation with Large Language Models," *IEEE Transactions on Neural Networks and Learning Systems*, 2024, doi: 10.1109/TNNLS.2024.XXXXXXX.

[9] S. Geng, M. Josifoski, M. Peyrard, and R. West, "Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning," *arXiv preprint arXiv:2305.13971*, 2023.

[10] Y. Liu, K. Zhang, and J. Sun, "StructPrompt: Structured Prompting for Deterministic Code Generation," in *Proc. IEEE Int. Conf. on Software Analysis, Evolution and Reengineering (SANER)*, 2024.

[11] R. Pan, T. Zhang, and M. Zhou, "On the Limits of Automated Feedback Loops for Code Generation with Large Language Models," in *Proc. EMNLP*, 2023.

[12] J. Pineau, P. Vincent-Lamarre, K. Sinha, et al., "Improving Reproducibility in Machine Learning Research," *J. Mach. Learn. Res.*, vol. 22, no. 164, pp. 1–20, 2021.

[13] M. Chen, J. Tworek, H. Jun, et al., "Evaluating Large Language Models Trained on Code," *arXiv preprint arXiv:2107.03374*, 2021.

[14] Y. Liu, J. Wang, and C. Zhai, "Evaluating Robustness of Automated Code Generation Systems," in *Proc. IEEE Int. Conf. on Software Analysis, Evolution and Reengineering (SANER)*, 2023.

[15] T. Wu, D. Khashabi, E. Kamar, and D. S. Weld, "Large language models as evaluators," *arXiv preprint arXiv:2307.03109*, 2023.