PLC Error Classification System — Evaluation Summary

Reformatted webpage — full contents preserved

Rabie Sofany / Rabie.sofany@gmail.com

> “The system models the full PLC build pipeline and classifies errors at the stage where the build actually fails. When later stages are not reached, this is explicitly reflected. Downstream failure messages are attached as context rather than misclassified as independent errors.”
>
> “Solid system design. Needs tighter precision in edge cases, but fundamentally correct.”

Objective

- Parse raw PLC build logs
- Identify and classify errors by severity, pipeline stage, and fix complexity
- Detect cascading failures
- Provide actionable, non-hallucinatory fix suggestions with code examples

Evaluation Scope & Methodology

- 30 representative PLC build logs were generated to cover XML, IEC semantic,
- For each log:
- Validation focused on:

Synthetic error generator prompt

```
Environment : Open AI GPT-5.2
Prompt for generation:
You are an expert PLC and industrial automation engineer.
Task

Generate 30 realistic PLC build/compilation log samples for an OT customer environment.
These logs will be used to test an AI system that parses logs, classifies the primary/root error,
and suggests fixes across a multi-stage pipeline:
xml_validation → code_generation → iec_compilation → c_compilation
```

```
Output Requirements (STRICT)
1) Output ONLY a CSV file as plain text (no markdown, no commentary).
2) Use UTF-8 compatible text only.
3) The CSV must have exactly these columns (in this order):
sample_id,title,raw_log
4) sample_id must be integers 1..30.
5) title must be a short human-readable label.
6) raw_log must contain the full multi-line log, with newlines encoded as \n (literal backslash-n)
so the CSV remains one row per sample.
7) Do NOT include any extra columns.
Log Realism & Formatting (STRICT)
- Logs must resemble real CLI output from PLC tooling (e.g., Beremiz/matiec-style), with
timestamps like [HH:MM:SS].
- Include realistic messages such as:
- "Building project…"
- "Cannot build project."
- "Generating SoftPLC IEC-61131 ST/IL/SFC code…"
- "Compiling IEC Program into C code…"
- Tool invocations (e.g., /root/beremiz/matiec/iec2c …)
- Optional gcc/ld output for C compilation stage
- When a stage is reached, the log must show at least one line that clearly indicates that stage
(e.g., "Generating SoftPLC…" implies code_generation reached).
- Use consistent PLCopen namespace strings when XML is involved:
http://www.plcopen.org/xml/tc6_0201
Coverage Requirements (STRICT)
Create exactly 30 logs distributed as follows:
A) 8 logs where the PRIMARY/root cause is xml_validation (XSD schema issues). The pipeline
should stop early (no IEC compilation lines).
B) 8 logs where the PRIMARY/root cause is code_generation (tool crash / Python traceback /
generator failure). Must include a Python traceback and an exception (e.g.,
AttributeError/TypeError).
C) 8 logs where the PRIMARY/root cause is iec_compilation (IEC semantic/syntax errors in
plc.st with line numbers, e.g., CONST assignment, type mismatch, undeclared var).
D) 6 logs where the PRIMARY/root cause is c_compilation (gcc/ld errors after IEC->C
succeeds, e.g., undefined reference, missing header, compile error in generated C).

Design Constraints (to avoid trivial repetition)
- No two logs may share the exact same primary error message string.
- Vary line numbers, program names (program0/program1), and file paths
(/tmp/.tmpXXXX/build/…).
- For iec_compilation logs, include at least one of:
- "Bailing out!"
- "1 error(s) found."
- "iec2c … exited with status 1"
- For c_compilation logs, include plausible gcc/ld output and indicate the C file name/location.
- Some logs may include earlier non-blocking warnings (e.g., an XML schema warning) but the
root cause must be the requested stage for that sample category. The log content must clearly
show which stage actually fails.
Safety / Policy (STRICT)
- Do NOT include any personal data, credentials, API keys, or real customer names.
- No external links.
Final Check (STRICT)
Before outputting the CSV, verify:
- Exactly 30 rows (sample_id 1..30)
- Exactly 3 columns
- raw_log uses \n for newlines
- Category counts match: xml_validation=8, code_generation=8, iec_compilation=8,
c_compilation=6
Now output the CSV only.
```

Ground truth generation prompt

```
Prompt: PLC Error Classification, Root Cause & Fix Table Generator
You are an expert PLC and industrial automation engineer with deep experience in
PLCopen XML, IEC-61131-3 toolchains, and PLC build pipelines.
Analyze the following PLC build or compilation log and produce a single, structured
Markdown table.
Objectives

From the log:
1. Identify all real errors and relevant warnings (do not invent issues).
2. Map each issue to the exact pipeline stage(s) where it occurs.
3. Classify the error type, severity, and fix complexity.
4. Explain why the error occurred (root cause) in engineering terms.
5. Provide 1–3 actionable fix suggestions per error, each with:
○ Clear before / after code snippets
○ A confidence score (0–1) indicating likelihood of resolving the issue
Allowed Pipeline Stages (use only these)
● xml_validation
● code_generation
● iec_compilation
● c_compilation
Include only stages that are actually reached according to the log.
Error Type Taxonomy
Use the most appropriate category:
● Syntax / Schema validation errors
● Logic errors
● Runtime errors
● Resource limits

● Hardware faults
○ I/O module failure
○ Power supply problems
○ Overheating
○ Memory corruption
● Communication failures
○ Network issues
○ Device conflicts
Severity Levels
● blocking – build or execution cannot proceed
● warning – non-fatal but risky or non-compliant
● info – informational only
Fix Complexity Levels
● trivial – small correction, config/schema fix
● moderate – localized logic or model changes
● complex – architectural or multi-component changes
Output Requirements (STRICT)
● Output only one Markdown table

● Do not include explanations outside the table
● Do not include stages that were not reached
● Use exact line numbers from the log when available
● Be concise, technical, and engineer-grade
Table Columns (MANDATORY ORDER)
| Error Type | Stage | Line Numbers | Severity | Fix Complexity | Root cause (why it occurred) |
Actionable suggestions (with confidence 0–1) |
Actionable Suggestions – Formatting Rules
● Provide 1–3 suggestions per error
● Each suggestion must include:
○ Confidence score (0–1)
○ Before code
○ After code
● Code must be fully shown, properly formatted, and readable
● Preserve realistic XML / JSON / IEC-61131 formatting and newlines
```

Results Overview

- Severity, Stage, and Complexity were correct in the majority of cases

- IEC semantic failures (e.g., CONST assignment violations) were classified with
- XML schema warnings were generally identified correctly as non-blocking
- Parsed error events are now cleanly consolidated, with downstream “build failed”
- Earlier issues with duplicated stages and artificial “BuildFailure” events have been
- Some logs still show over-representation of consequence information in internal
- C compilation stages (gcc, ld) are not yet fully parsed and may fall back to unknown
- has_cascading_errors correctly reflects the presence of multiple issues
- Individual errors are no longer automatically marked as cascading without explicit
- Suggestions are generally actionable and relevant
- IEC semantic violations are handled with accurate corrective patterns
- XML guidance is practical and schema-aware
- In a minority of cases, root-cause targeting drifts (e.g., wrong variable named)
- Some suggestions are duplicated across related events
- Occasional speculative explanations appear when tool-internal context is missing

Key Strengths

- Correct mental model of a real PLC build pipeline
- Clear separation between root causes and consequences
- Deterministic, explainable classification behavior
- Robust handling of common industrial error patterns
- Output is structured, auditable, and suitable for automation

Known Limitations (Explicitly Acknowledged)

- Native C compilation errors are not fully supported yet
- Severity inflation may occur for non-blocking XML warnings
- Fix suggestion accuracy depends on parser precision and available context

Overall Assessment

- Solid architectural thinking
- Accurate PLC domain understanding
- Practical AI integration without over-automation
- Clear awareness of pipeline semantics and error causality

Final Verdict

✅ Requirement met. Strong technical exercise with clear, explainable behavior and realistic trade-offs.
Primar Cascadi
Severity Stage Complexity y Internal ng Suggestio Suggestio
(Expected (Expected (Expected → Reaso Classif Structur Handlin ns – Root ns –

Per-log Evaluation Table (Logs 1–30)

Note: This table is reproduced from the original report. Line breaks inside cells are preserved in the Notes column for readability.

Log	Severity (Expected → Output)	Stage (Expected → Output)	Complexity (Expected → Output)	Reasoning	Primary Classification	Internal Structure	Cascading Handling	Suggestions – Root Cause	Suggestions – Quality	Notes
1	blocking → blocking ✅	iec_compilation → iec_compilation ✅	trivial → trivial ✅	✅	3/3 ✅	❌	❌	❌	⚠	Extra stages previously existed; fix targets wrong var in one suggestion
2	warning → warning ✅	xml_validation → xml_validation ✅	trivial → trivial ✅	✅	3/3 ✅	✅	✅	✅	✅	Clean XML schema classification
3	blocking → blocking ✅	code_generation →	moderate → moderate ✅	✅	3/3 ✅	⚠	⚠	⚠	⚠	Traceback handled, but
4	blocking → blocking ✅	iec_compilation → iec_compilation ✅	trivial → trivial ✅	✅	3/3 ✅	❌	❌	❌	⚠	Consequence messages duplicated as separate events
5	blocking → warning ❌	xml_validation → xml_validation ✅	trivial → trivial ✅	⚠	2/3 ⚠	✅	✅	⚠	⚠	Severity inflated; pipeline actually proceeds

Log	Severity (Expected → Output)	Stage (Expected → Output)	Complexity (Expected → Output)	Reasoning	Primary Classification	Internal Structure	Cascading Handling	Suggestions – Root Cause	Suggestions – Quality	Notes
6	blocking → blocking ✅	c_compilatio n → unknown ❌	moderate → moderate ✅	❌	1/3 ❌	❌	❌	⚠	❌	C compiler stage not detected
7	blocking → blocking ✅	iec_compilation → iec_compilation ✅	trivial → trivial ✅	✅	3/3 ✅	✅	✅	✅	✅	Strong, canonical IEC semantic case
8	warning → blocking ❌	xml_validation → xml_validation ✅	trivial → trivial ✅	⚠	2/3 ⚠	✅	✅	⚠	⚠	XML treated as blocking though pipeline continues
9	blocking → blocking ✅	code_generation → code_generation ✅	moderate → complex ❌	⚠	2/3 ⚠	⚠	⚠	⚠	⚠	Complexity inflated without architectural evidence
10	blocking → blocking ✅	iec_compilation → iec_compilation ✅	trivial → trivial ✅	✅	3/3 ✅	❌	❌	❌	⚠	Fix duplicated across consequence events
11	blocking → blocking ✅	c_compilatio n → unknown ❌	moderate → moderate ✅	❌	1/3 ❌	❌	❌	⚠	❌	gcc/ld patterns not parsed
12	warning → warning ✅	xml_validation → xml_validation ✅	trivial → trivial ✅	✅	3/3 ✅	✅	✅	✅	✅	Clean

Log	Severity (Expected → Output)	Stage (Expected → Output)	Complexity (Expected → Output)	Reasoning	Primary Classification	Internal Structure	Cascading Handling	Suggestions – Root Cause	Suggestions – Quality	Notes
13	blocking → blocking ✅	iec_compilation → iec_compilation ✅	trivial → trivial ✅	✅	3/3 ✅	⚠	⚠	❌	⚠	Wrong variable named in one fix
14	blocking → blocking ✅	code_generation →	moderate → moderate ✅	✅	3/3 ✅	⚠	⚠	⚠	⚠	Root correct, but generator cause
15	warning → blocking ❌	xml_validation → xml_validation ✅	trivial → trivial ✅	⚠	2/3 ⚠	✅	✅	⚠	⚠	Severity over-classified
16	blocking → blocking ✅	iec_compilation → iec_compilation ✅	trivial → trivial ✅	✅	3/3 ✅	❌	❌	❌	⚠	Consequence duplication again
17	blocking → blocking ✅	c_compilatio n → unknown ❌	moderate → trivial ❌	❌	0/3 ❌	❌	❌	❌	❌	C pipeline unsupported
18	blocking → blocking ✅	code_generation → code_generation ✅	moderate → moderate ✅	✅	3/3 ✅	⚠	⚠	⚠	⚠	Acceptable for MVP
19	warning → warning ✅	xml_validation → xml_validation ✅	trivial → trivial ✅	✅	3/3 ✅	✅	✅	✅	✅	Clean
20	blocking → blocking ✅	iec_compilation → iec_compilation ✅	trivial → trivial ✅	✅	3/3 ✅	❌	❌	❌	⚠	Duplicate fixes

Log	Severity (Expected → Output)	Stage (Expected → Output)	Complexity (Expected → Output)	Reasoning	Primary Classification	Internal Structure	Cascading Handling	Suggestions – Root Cause	Suggestions – Quality	Notes
21	blocking → blocking ✅	c_compilatio n → unknown ❌	moderate → moderate ✅	❌	1/3 ❌	❌	❌	⚠️	❌	No C-stage regex
22	warning → warning ✅	xml_validation → xml_validation ✅	trivial → trivial ✅	✅	3/3 ✅	✅	✅	✅	✅	Clean
23	blocking → blocking ✅	iec_compilation → iec_compilation ✅	trivial → trivial ✅	✅	3/3 ✅	⚠️	⚠️	❌	⚠️	Wrong variable again
24	blocking → blocking ✅	code_generation → code_generation ✅	moderate → moderate ✅	✅	3/3 ✅	⚠️	⚠️	⚠️	⚠️	Acceptable
25	warning → blocking ❌	xml_validation → xml_validati	trivial → trivial ✅	⚠️	2/3 ⚠️	✅	✅	⚠️	⚠️	Severity inflation
26	blocking → blocking ✅	iec_compilation → iec_compilation ✅	trivial → trivial ✅	✅	3/3 ✅	❌	❌	❌	⚠️	Consequence duplication
27	blocking → blocking ✅	c_compilatio n → unknown ❌	moderate → complex ❌	❌	0/3 ❌	❌	❌	❌	❌	C stage unsupported
28	warning → warning ✅	xml_validation → xml_validation ✅	trivial → trivial ✅	✅	3/3 ✅	✅	✅	✅	✅	Clean

Log	Severity (Expected → Output)	Stage (Expected → Output)	Complexity (Expected → Output)	Reasoning	Primary Classification	Internal Structure	Cascading Handling	Suggestions – Root Cause	Suggestions – Quality	Notes
29	blocking → blocking ✅	iec_compilation → iec_compilation ✅	trivial → trivial ✅	✅	3/3 ✅	⚠	⚠	❌	⚠	Wrong variable once more
30	blocking → blocking ✅	code_generation → code_generation ✅	moderate → moderate ✅	✅	3/3 ✅	⚠	⚠	⚠	⚠	Acceptable

Scoring Method

- ✅ = 100%
- ⚠ = 66%
- ❌ = 0%
- Numeric scores (e.g., 3/3, 2/3, 1/3, 0/3) mapped linearly to %
- Total rows = 30

Column-wise Performance

Severity (Expected → Output)	83.3%
Stage (Expected → Output)	76.7%
Complexity (Expected → Output)	86.7%
Reasoning	80.7%
Primary Classification	84.4%
Internal Structure	53.3%

Cascading Handling	**53.3%**
Suggestions – Root Cause	**43.3%**
Suggestions – Quality	**63.3%**

What This Tells You (Executive Read)

- Severity classification
- Complexity estimation
- Primary classification correctness
- Reasoning consistency
- Stage detection
- Suggestion quality
- Internal structure
- Cascading error handling
- Root-cause suggestions

Root Cause of Low Scores (Systemic)

1. C compilation unsupported
○ No gcc/ld regex → stage = unknown
○ Cascades into:
■ Wrong stage
■ Wrong complexity
■ Poor suggestions
2. Consequence duplication
○ Same root issue emitted multiple times
○ Hurts internal structure + cascading metrics
3. Speculative fixes
○ Correct symptom, wrong variable or architectural leap
○ Penalizes root-cause accuracy

System Enhancements

- Add C compiler pattern detection
- Collapse duplicate consequence events

- Introduce root-cause anchoring
- Enforce 1 root cause → N consequences

Bottom Line

Classifier accuracy is strong (≈85%).
Structural intelligence and causal reasoning are the current bottleneck (≈50%).
Fixing C-stage parsing and consequence deduplication will unlock the next quality tier.
