
Trust, but verify

Keeping AI-generated embedded code correct, safe
and compliant

SPEAKER

Dr. Marc Thomas

Field Application Engineer
IAR Systems (Qt Group)

SOURCE

Presented at the Embedded Software & AI seminar, WoTS 2026 · Jaarbeurs Utrecht, the Netherlands
· 22 September 2026
Slides © Dr. Marc Thomas, IAR Systems (Qt Group)

Trust, but Verify: Keeping AI-Generated Embedded Code Correct, Safe and Compliant

AI assistants have moved from suggesting lines to acting on the codebase: today's agentic tools write, refactor, and run tests on their own. They can compress hours of embedded C/C++ work into minutes, but they raise an unavoidable question for any team shipping safety- or security-critical firmware: can you trust what the agent produced, and can you prove it?

This session looks at where productivity meets accountability. Starting from a realistic AI-assisted workflow, it shows how to keep verification in the loop rather than bolted on at the end: static analysis to catch defects and coding-standard violations early, runtime checking to expose errors that surface only during execution, automated testing for fast regression feedback, and MISRA/CERT compliance treated as a continuous gate. Examples use IAR's analysis and verification tooling (C-STAT, C-RUN) to make the principles concrete.

For teams developing to ISO 26262, IEC 62304 or IEC 61508, the same discipline — backed by a certified compiler and the application of its functional-safety guide — is what lets AI-assisted code stand up inside a safety case.

Agenda

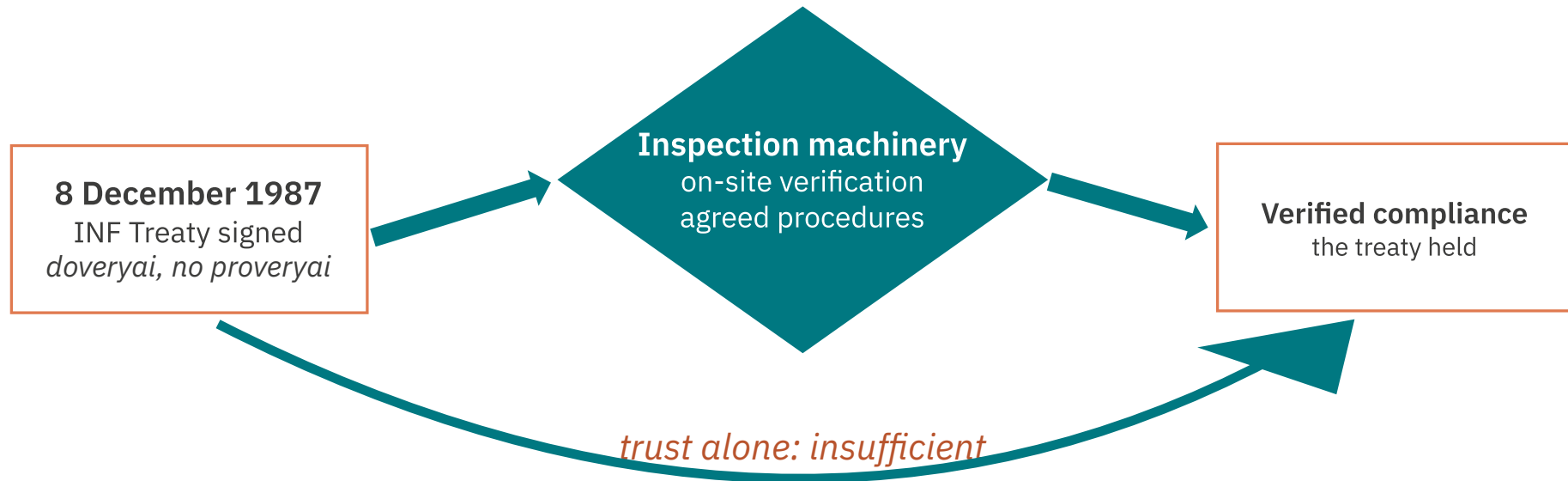
-
- 01 Introduction: the Reagan paradox
 - 02 Act I — the metric of distrust
 - 03 Act II — the evidence under the microscope
 - 04 Act III — the structural failure
 - 05 Act IV — the five-gate architecture
 - 06 Act V — the safety case
 - 07 Conclusion: Monday morning
-

Introduction:

the Reagan paradox

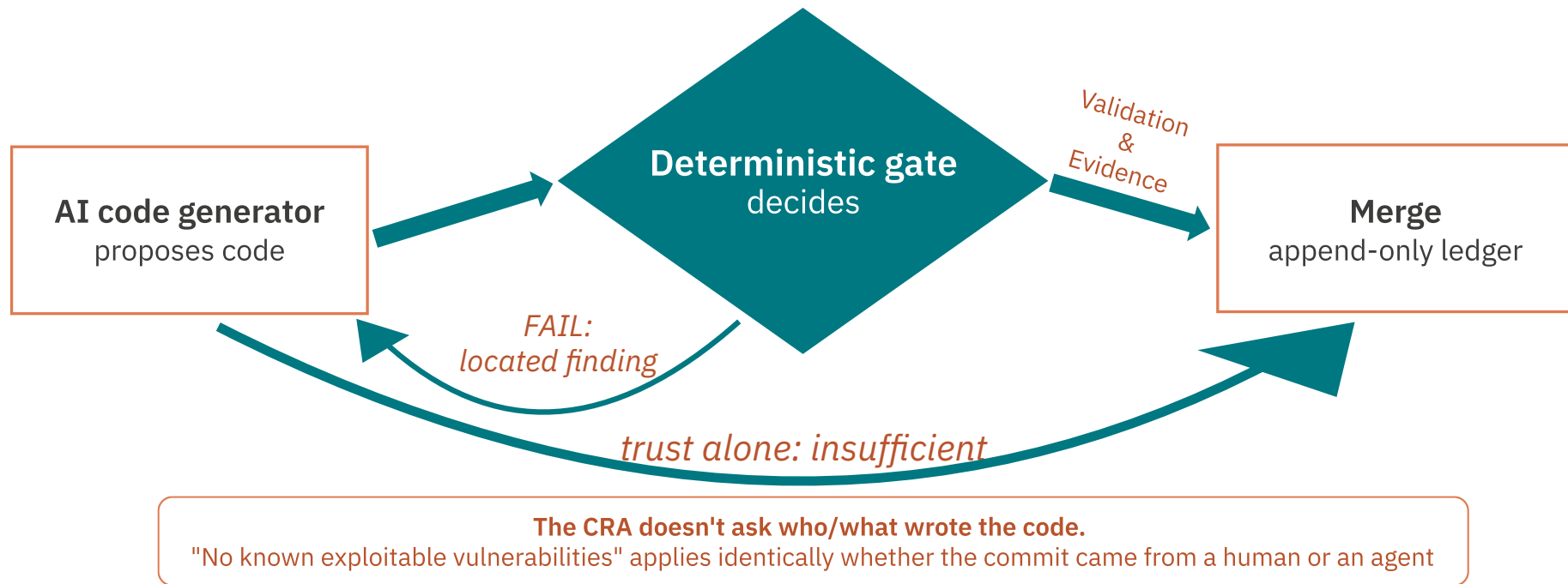
The Reagan paradox : Verification was the treaty

- 8 December 1987: Reagan signs the INF (Intermediate-Range Nuclear Forces) Treaty quoting “**dove**ryai, no **pro**veryai” → **trust, but verify**.
- The treaty held because of its inspection machinery — on-site verification and agreed procedures — **not interpersonal trust**.
- Same move for AI-generated code: **inspect the machinery of verification, not the sincerity of the generator**.



The AI agent proposes and the gate decides

- **AI agents change the rate at which code arrives.** They do not change the safety requirements of the gate.
- **The generator is probabilistic;** the gate must stay deterministic to carry compliance and regulatory weight.
- In the age of **probabilistic** generation, **verification is the trust.**

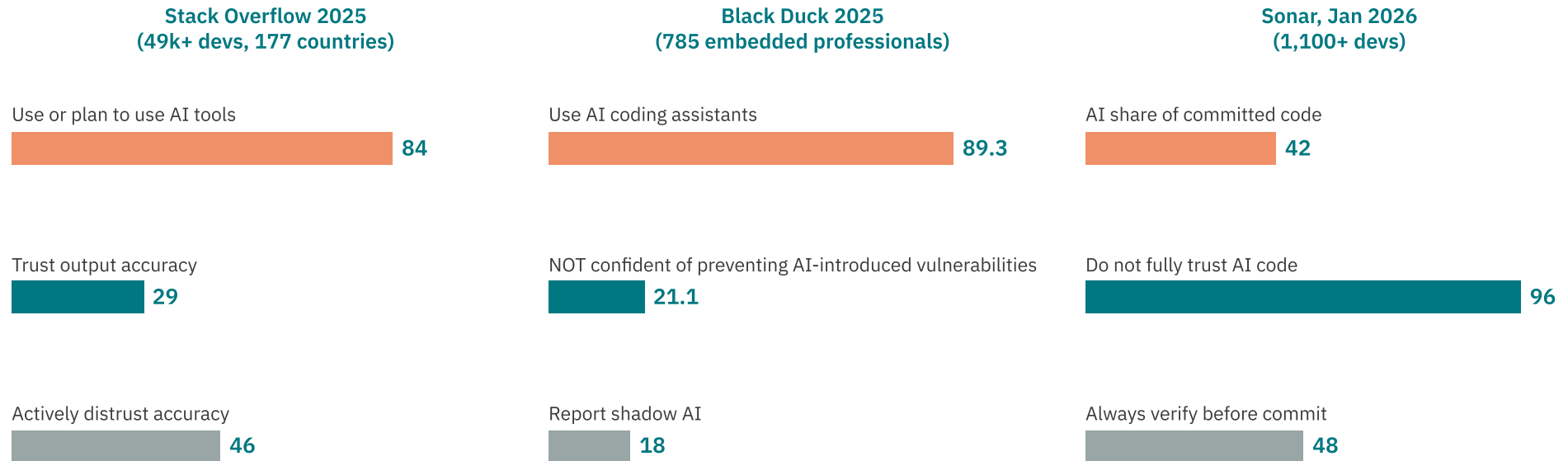


Act I:

the metric of distrust

Adoption is near-universal; assurance is not

- Embedded adoption (89.3%) is higher than the general developer figure (84%) — safety-critical teams are not holding back.
- The figures below answer different survey questions:

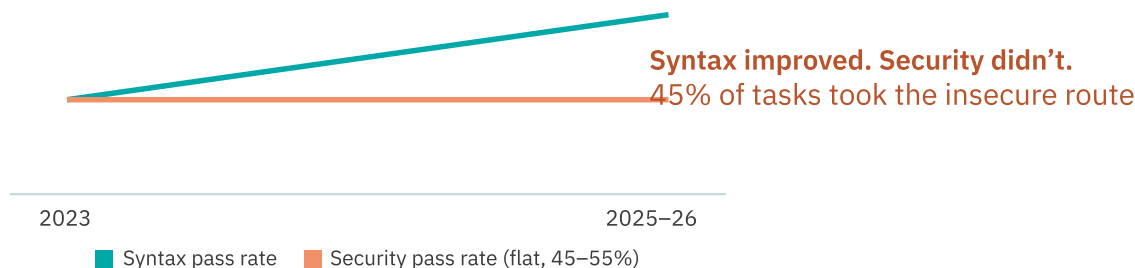


Stack Overflow, 2025 Developer Survey — AI section (49,000+ respondents, 177 countries). survey.stackoverflow.co/2025/ai

Black Duck 2025 (Censuswide, n = 785, June 2025) · Sonar, State of Code Developer Survey, Jan 2026 (n = 1,100+) — vendor research; cite the method with the number.

Insecure defaults, shadow AI

- **Insecure by default:** The syntax pass is now $\approx 95\%$, security pass still $\approx 45\text{--}55\%$. (Veracode 2025)



- **Shadow AI :** 18% of embedded organizations report unsanctioned use of shadow AI; (Black Duck, 785 professionals, June 2025.)



Veracode, “2025 GenAI Code Security Report” (Jul 2025) “2025_GenAI_Code_Security_Report_Final.pdf”

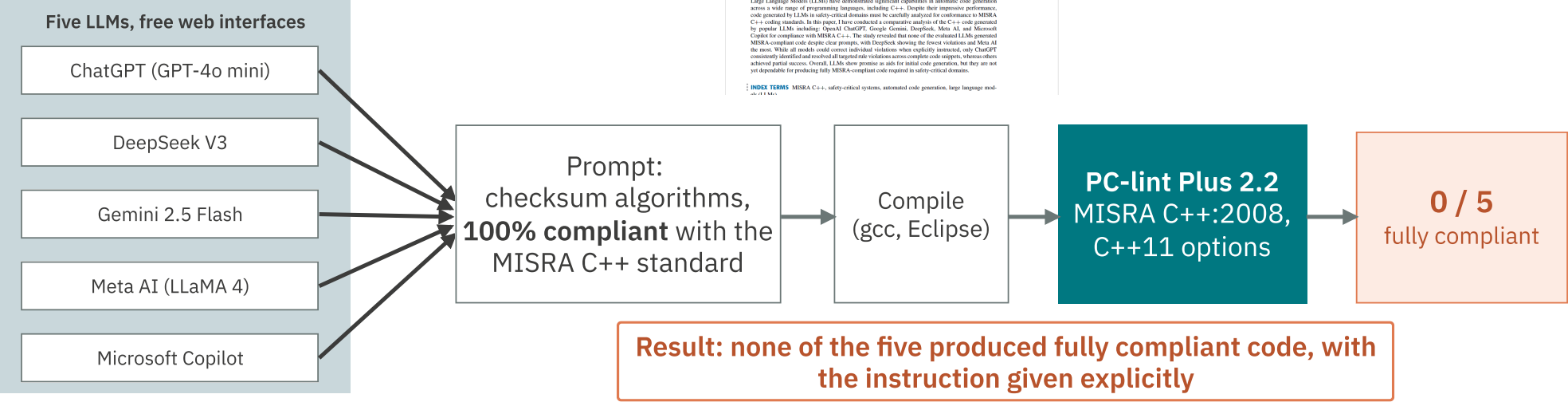
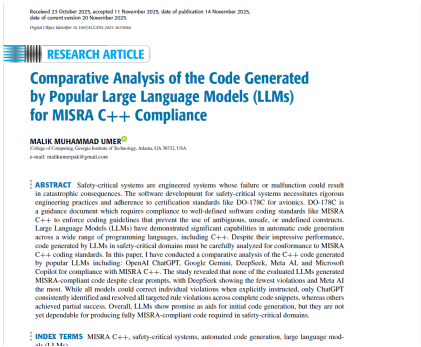
Black Duck, THE STATE OF EMBEDDED SOFTWARE QUALITY AND SAFETY 2025” (state-of-embedded-software-quality-and-safety.pdf) (shadow-AI figures).

Act II:

the evidence under the microscope

The anchor study: five models, one instruction, zero compliant

M. M. Umer, “Comparative Analysis of the Code Generated by Popular Large Language Models (LLMs) for MISRA C++ Compliance,”

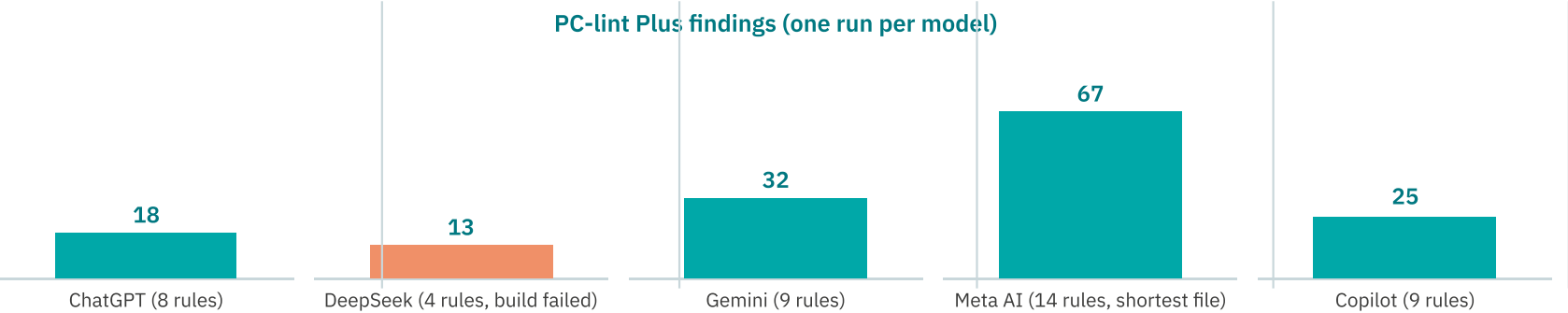


M. M. Umer, “Comparative Analysis of the Code Generated by Popular Large Language Models (LLMs) for MISRA C++ Compliance,” IEEE Access, vol. 13, pp. 194815–194831, 2025. doi:10.1109/ACCESS.2025.3633086

What failed: ordinary rules, not exotic ones

- Three ordinary rules carry 86 of the 155 total findings
- One run per model, so no variance data; per-model counts are absolute, not normalised

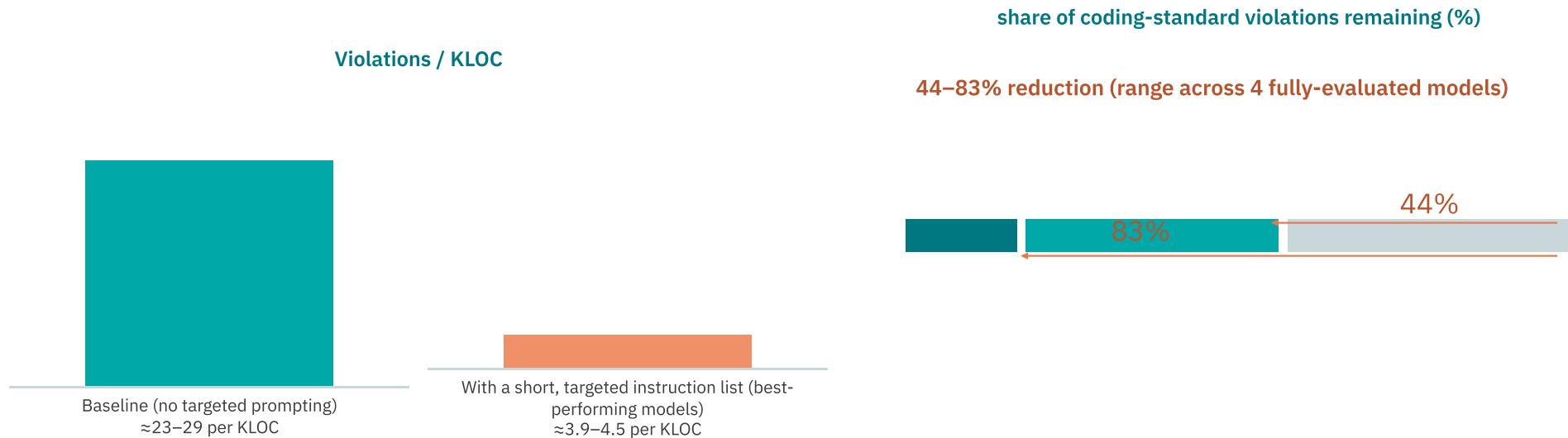
Finding	Total	ChatGPT	DeepSeek	Gemini	Meta AI	Copilot
Signedness	44	8	5	10	21	0
Missing const	27	0	6	7	7	7
Bitwise-on-signed	15	0	0	6	9	0
All other rules	69	10	2	9	30	18



M. M. Umer, IEEE Access, vol. 13, 2025. doi:10.1109/ACCESS.2025.3633086 (CC BY 4.0)
Per-model figures: the paper’s Tables 1–2; the 155 / 20 totals are the author’s arithmetic (Derived); repositories return 404.

Prompting helps — and is not a safety case

- Without any targeted prompting, these models made about 23–29 rule violations for every 1,000 lines of code they wrote.
 - Giving each model a short list of its own most common mistakes (just 3 to 10 rules) cut that by 44 to 83%.
 - A short, targeted list worked better than pasting in the entire coding standard.
- But in two of those cases, the code broke more often. Some of that improvement came at the cost of working functionality.
 - None of these tests got all the way to zero violations.
 - A safety case needs zero, with evidence behind it — not just fewer mistakes.

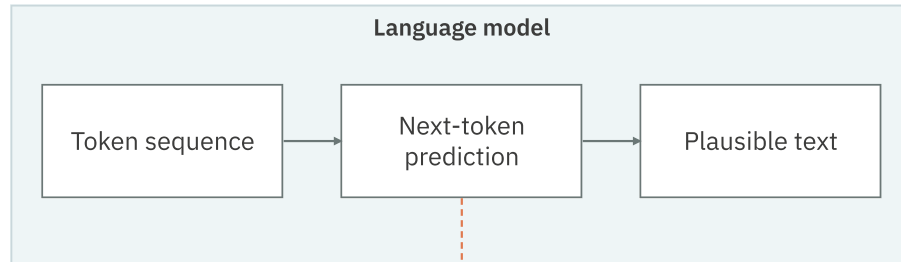


“Reducing MISRA violations in LLM-generated code,” Research Square preprint, 25 Nov 2025. doi:10.21203/rs.3.rs-8123173/v1

— Act III: the structural failure

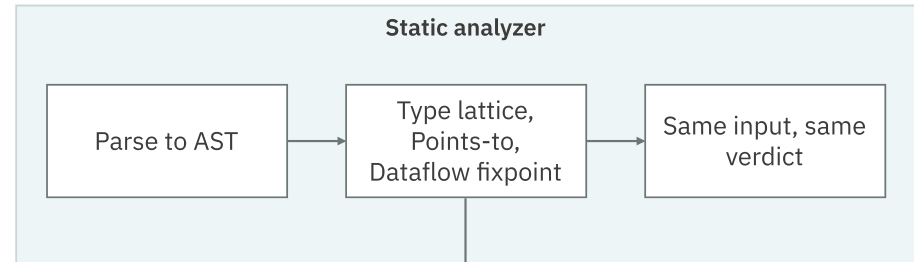
A text predictor cannot be its own oracle

- LLMs verdict varies run to run: a sample, not a **deterministic** answer.
- Independence and repeatability** are the whole value of an oracle.
- The analyser is the oracle**; the LLM model can sit upstream (triage) or downstream (draft deviation rationales); the model can never be the gate.



Verdict varies run to run:
a sample, not an answer

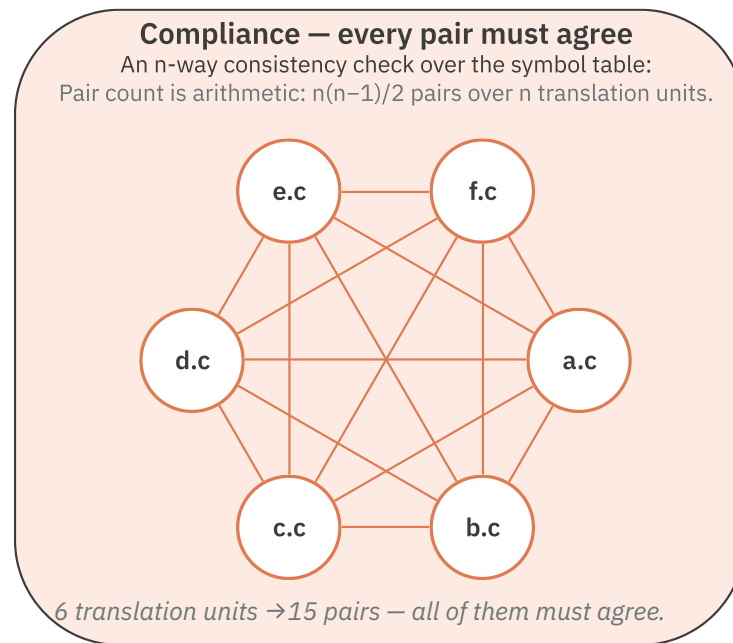
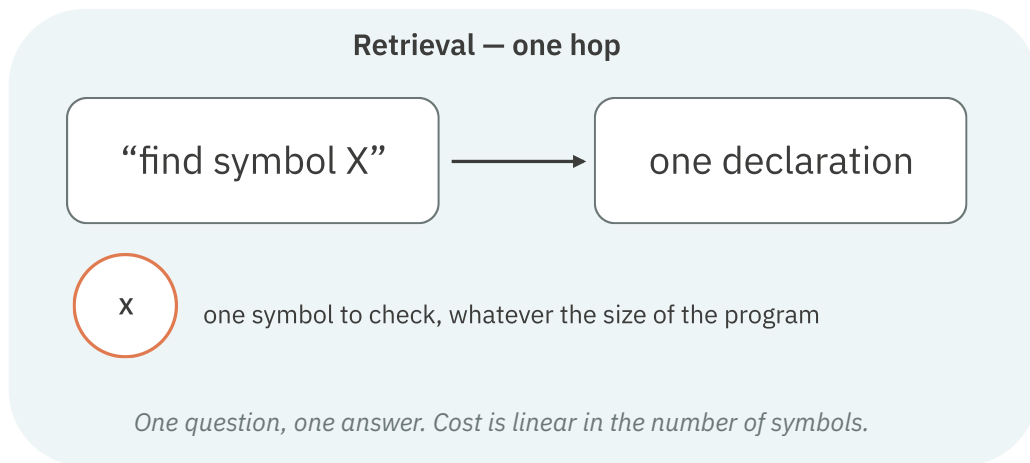
The LLM transformer architecture doesn't provide:
Type lattice,
No points-to abstraction
No dataflow fixpoint



The oracle

Compliance is an n-way join over the symbol table

- **Retrieval is one hop:** Find the declaration, return it.
- **Compliance is a join:** Every declaration must agree with every other one, in every translation unit.
- **A bigger AI context window improves retrieval. It does not convert a combinatorial verification into a lookup:**
 - Pairs grow quadratically as n^2
 - An agent's local scope hides exactly those collisions.



Retrieval Vs Compliance : two different operations.

- A bigger LLM context window improves retrieval, not verification
 - The window makes the lookup better. The join still has to be computed by a compiler, not by a predictor.

	Retrieval (lookup)	Compliance (join)
Question	“Where is symbol X?”	“Do all declarations of X agree?”
Logic	Locate one fact, one hop	n-way consistency check
Cost	Linear in the number of symbols	Quadratic: $n(n-1)/2$ pairs
Works on	Text inside the window	Symbol table for the whole program
Bigger window	Helps — more facts in reach	No help — the join still has to run

Firmware in tokens → no LLM context window is large enough

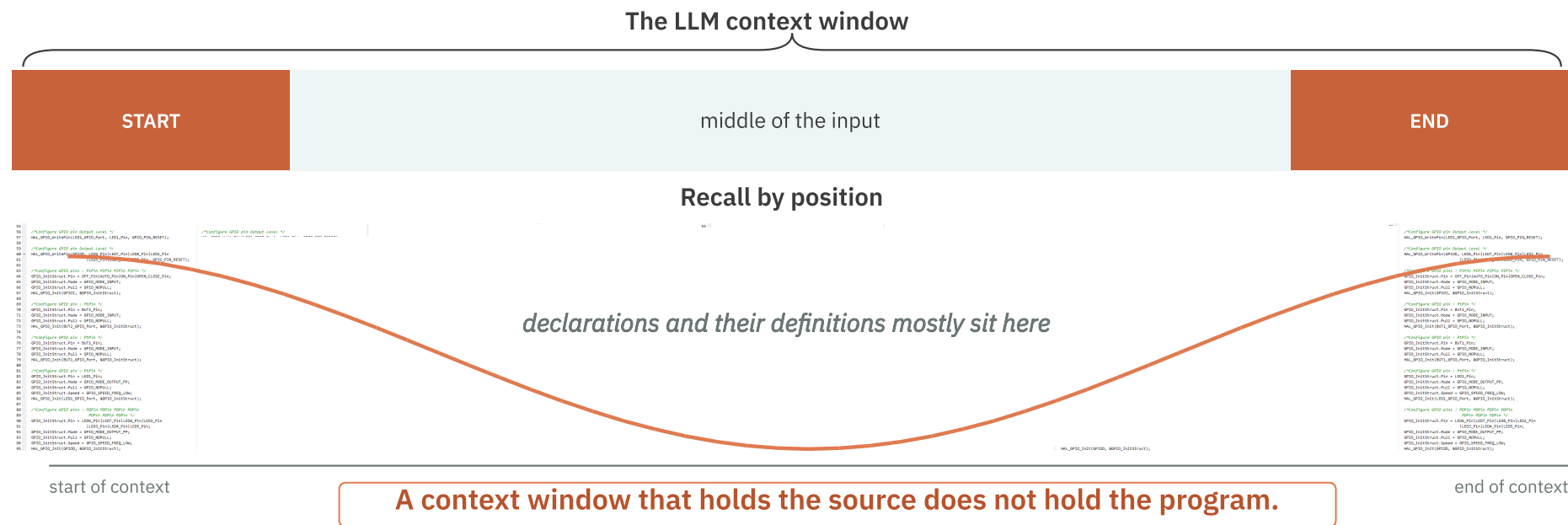
- **The multiplier everyone forgets:** analysis runs on preprocessed translation units → **vendor headers re-expand into every unit.**
- An analyser collapses the duplication into one symbol table; a context window pays for every copy.

Firmware class	Flash image	Source size	Source tokens	Whole-source review?
Sensor node, 8/16-bit	32–64 kB	8–20 kLOC	0.08–0.26 M	Yes — the least interesting row
Connected MCU + RTOS	256–512 kB	50–120 kLOC	0.5–1.6 M	At or beyond current windows
Rich MCU / MPU application	1–4 MB	150–500 kLOC	1.5–6.5 M	No
Automotive ECU class	4–16 MB	0.5–2 MLOC	5–26 M	No, by an order of magnitude

- Window sizing ignores the preprocessor multiplier — vendor headers re-expand into every translation unit.
- Recall degrades in the middle of long inputs, and a program has no privileged position (“lost in the middle”).

Lost in the middle (of the LLM context window), at firmware scale

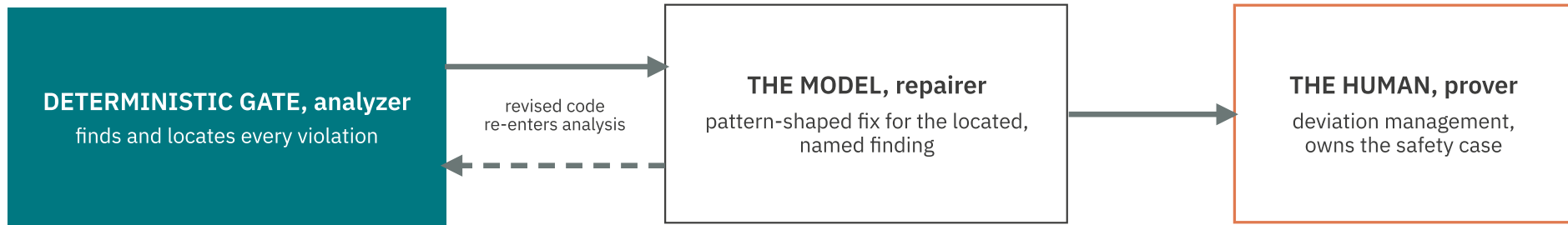
- Recall sags in the middle, and a program has no privileged position
- Even inside the AI context window, recall is strongest at the start and the end of the input and weakest in between.
- A perfect recall would not give the LLM a symbol table.



N. F. Liu et al, “Lost in the Middle,” Transactions of the ACL, vol. 12, pp. 157–173, 2024. Curve is schematic, not plotted data.

The asymmetry designs the workflow : locating is not applying

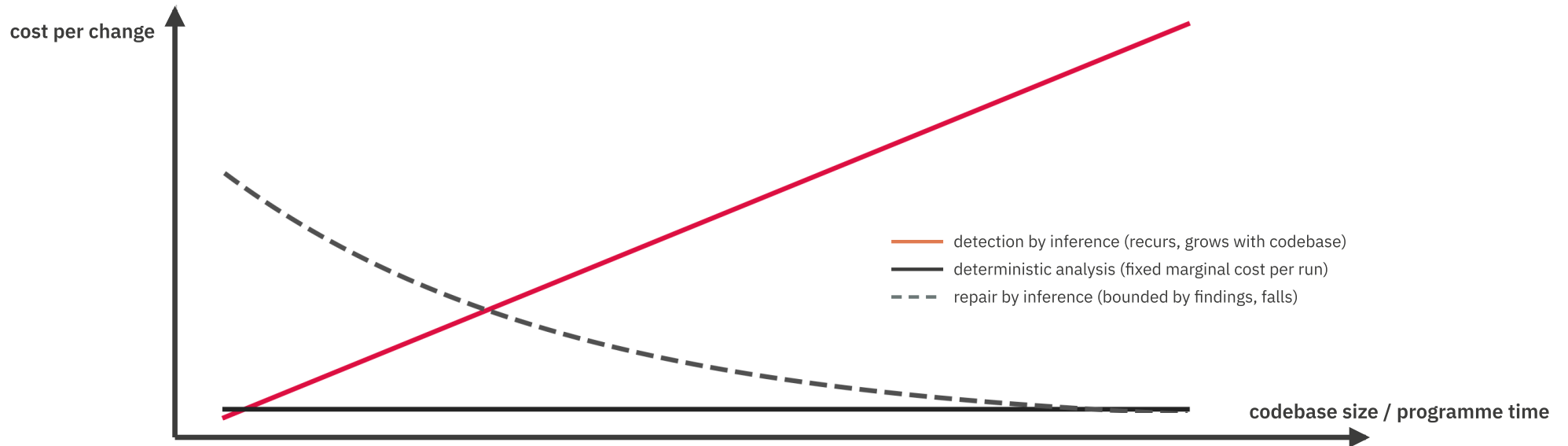
- **Informed (prompted) about a single, located, named violation, every LLM model fixed it;**
- **Locating is not applying:** The models was not able to enumerate all instances of a rule → performance collapsed (worst cell: 4 of 21 instances)
- **Division of labour:**
 - the deterministic gate enumerates
 - the model repairs
 - the human proves.



M. M. Umer, "Comparative Analysis of the Code Generated by Popular Large Language Models (LLMs) for MISRA C++ Compliance," IEEE Access, vol. 13, pp. 194815–194831, 2025.
doi:10.1109/ACCESS.2025.3633086; Tables 9–11 (guided-remediation experiment)

Why deterministic gates ? Spend tokens on repair, not detection

- Repair cost is bounded by findings and falls as quality improves.
- The analyser detects and the model repairs located findings



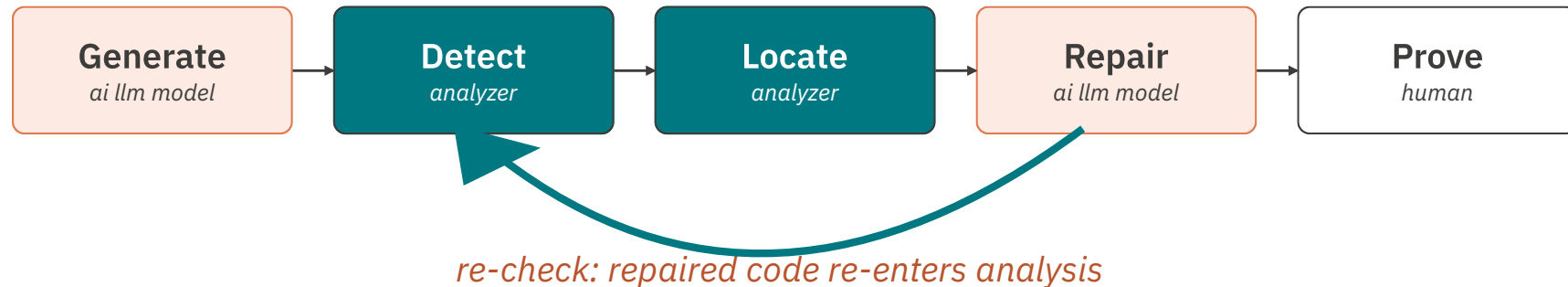
Allocate accordingly: let the fixed-cost analyzer locate every instance
AI LLM models are reliable at fixing a located and named defect, not at finding all of them.
Spend the metered AI agent on repairing what it finds!

Act IV:

the five-gate architecture

Five sub-tasks, three owners

- Producing compliant code is 5 tasks: **generate a candidate**, **detect that something is wrong**, **locate where**, **repair it**, **prove the result**.
 - The **model** owns generate
 - The **analyser** owns detect and locate — global.
 - The **model** repair: local, pattern-shaped.
 - Prove is a **human** evidence activity.



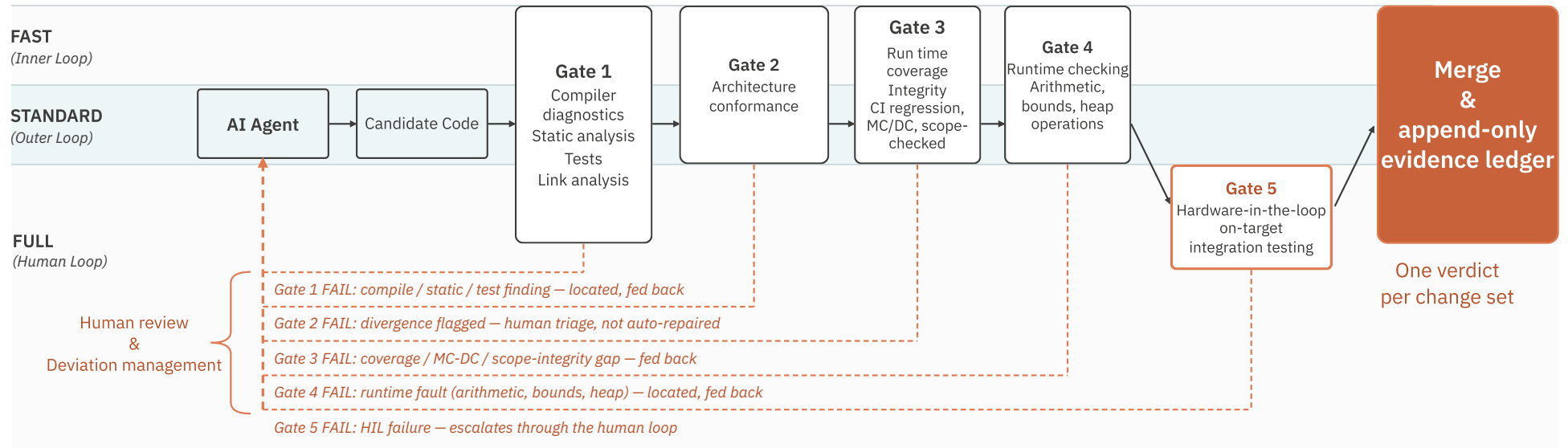
Prove is a human evidence activity !

A system that generates without an oracle is an unbounded search with no acceptance test.

The Analyzer → The five fused deterministic gates

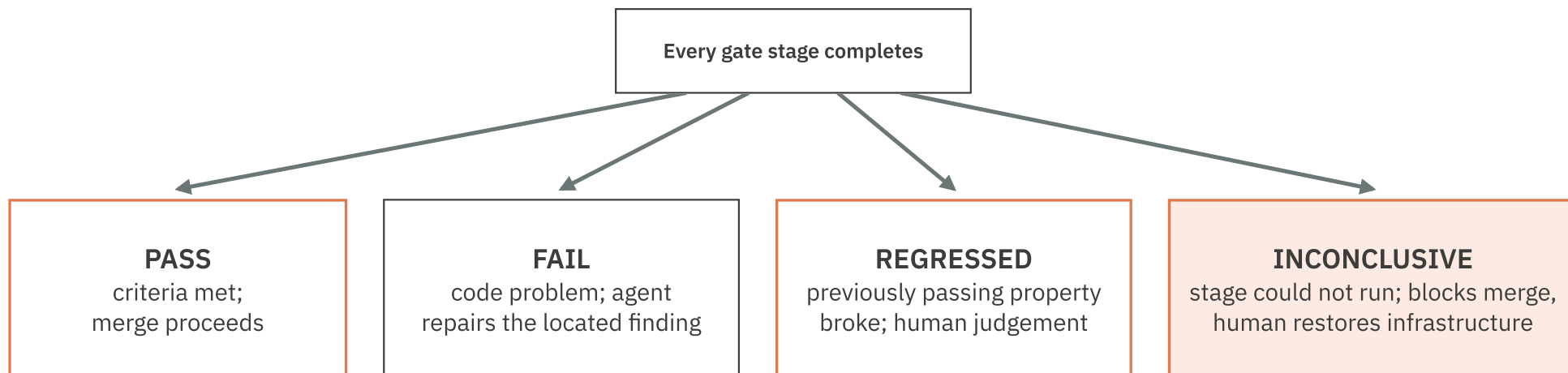
- AI changes the rate at which code arrives at the gate, but it changes nothing about what the gate must be.
- Productivity meets accountability. **Verification must be in the loop, not bolted on at the end.**
- **The agent participates inside the loop; the gates decide. FAIL returns a located repair payload — never a vague “try again”.**

The 5-Gate architecture required to make agentic code stand up in a safety case:



Deterministic gates outputs: Four machine-readable outcomes

The pipeline relies on 4 distinct outputs PASS, FAIL, REGRESSED, and INCONCLUSIVE outcomes to prevent agent manipulation.

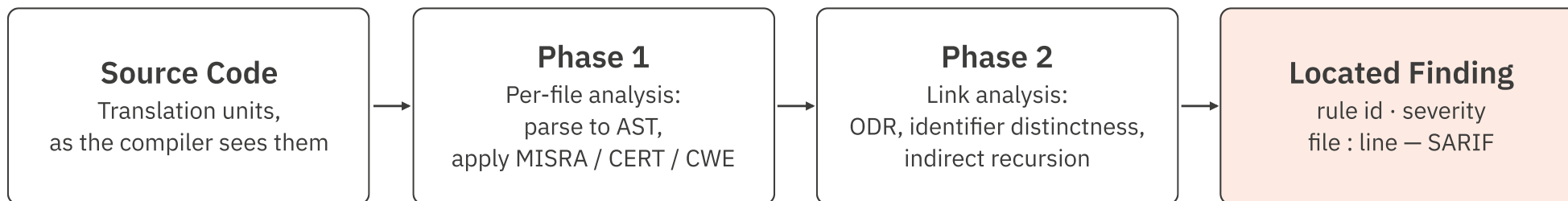


These machine-readable outcomes ensure the fast path never ships, and infrastructure failures never pass.

Gate 1: Static Analysis

Desktop/DevOps Server gate execution

- Parses source to an abstract syntax tree and checks it against a coding-standard rule set — the same input always gives the same verdict.
- Runs twice: per-file analysis catches local violations; a link-time pass catches whole-program issues no single file can see.
- Output is a located finding — rule ID, severity, file and line — never a vague warning.



A per-file fast path is legitimate mid-loop — and illegitimate as a release gate, because the link-phase findings only appear there.

AST — Abstract syntax tree. The parsed representation a static analyser checks against a coding-standard rule set

ODR (One Definition Rule) — The C++ rule that a link-time static-analysis pass checks for violations of, alongside identifier distinctness and indirect recursion; issues no single-file analysis pass can see.

IAR solution — C-STAT: MISRA C/C++, CERT and CWE-mapped static analysis, built into IAR Embedded Workbench. [iar.com/embedded-development-tools/iar-c-stat](https://www.iar.com/embedded-development-tools/iar-c-stat)

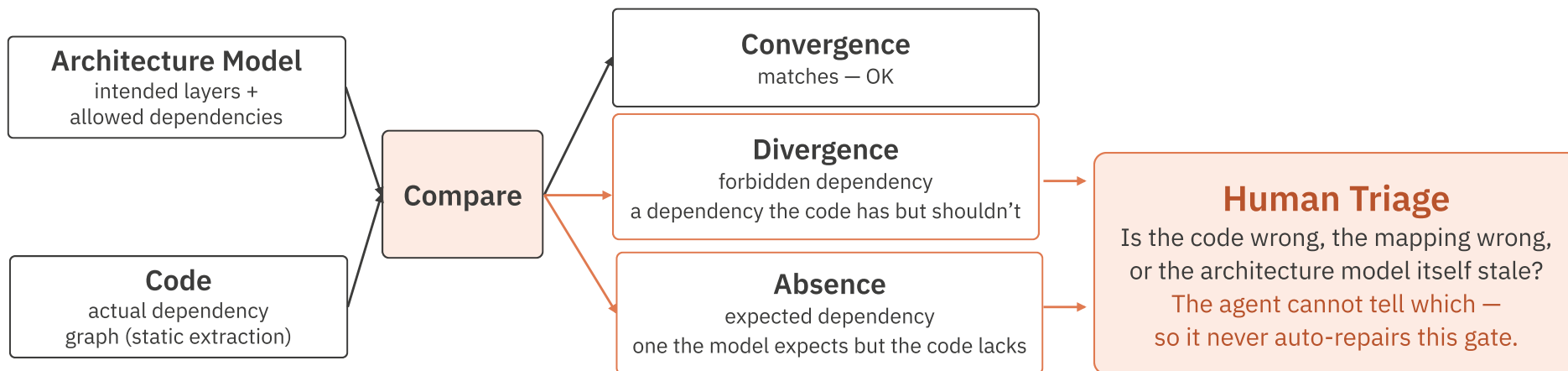
Qt Group solution — Axivion Suite: MISRA, CERT, AUTOSAR C++14 and CWE-mapped static analysis. [qt.io/product/quality-assurance/axivion-suite](https://www.qt.io/product/quality-assurance/axivion-suite)

Both solutions are TÜV-certified — ISO 26262 ASIL D, IEC 61508 SIL 4, IEC 62304 Class C, EN 50716 SIL 4 — with 100% automated MISRA and CWE rule coverage.

Gate 2: Architecture Conformance

Desktop/DevOps Server gate execution

- Checks that the code's actual structure still matches the architecture it was designed against — layers, allowed and forbidden dependencies.
- A finding has a code location only half the time: Divergence & Absence
- **Architecture should never auto-repaired → Human Triage**

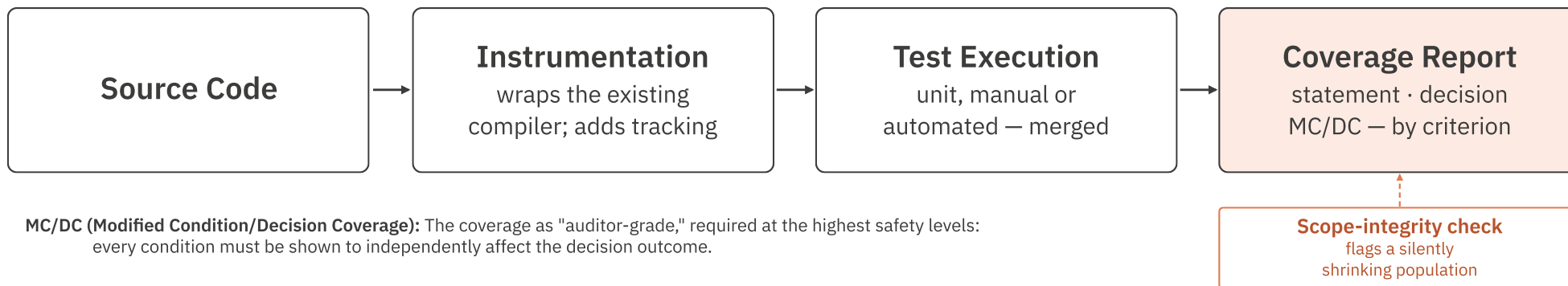


Qt Group solution: Axivion Suite: static and architecture-conformance analysis, clone/cycle detection. qt.io/product/quality-assurance/axivion-suite

Gate 3: Run time Coverage Integrity

Host/simulator in dev — on-target for safety-case evidence

- Instruments the code, runs the test suite, and measures exactly which statements, decisions and conditions actually executed.
- MC/DC is the auditor-grade criterion required at the highest safety levels — every condition shown to independently affect the outcome.
- A percentage alone isn't the evidence: the population it's computed over has to be checked too, or it can silently shrink.

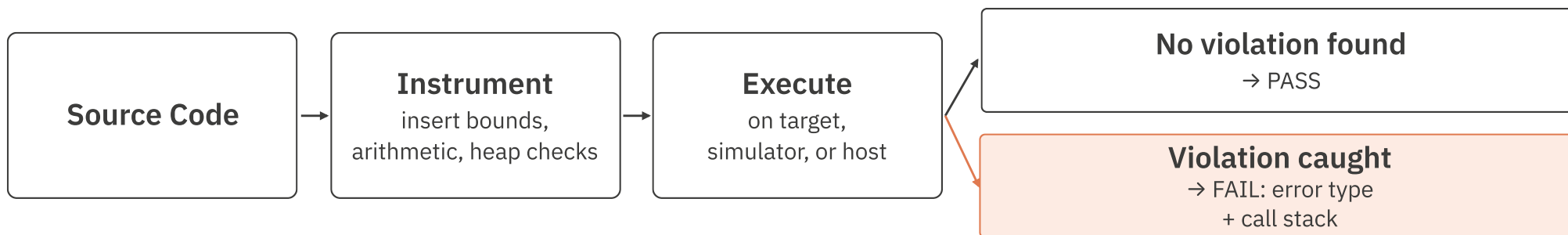


Qt Group solution: Coco: statement, decision, condition and MC/DC coverage, including on-target. qt.io/product/quality-assurance/coco

Gate 4: Runtime Checking

Host/simulator in dev — on-target for safety-case evidence

- Catches the errors static analysis can only suspect — a real buffer overflow or arithmetic overflow, on real data.
- Instruments the build so arithmetic, bounds and heap operations are checked live, during execution, with a call stack.
- Runs on the target, a simulator, or the host — static analysis reasons about the code; this reasons about its behaviour.

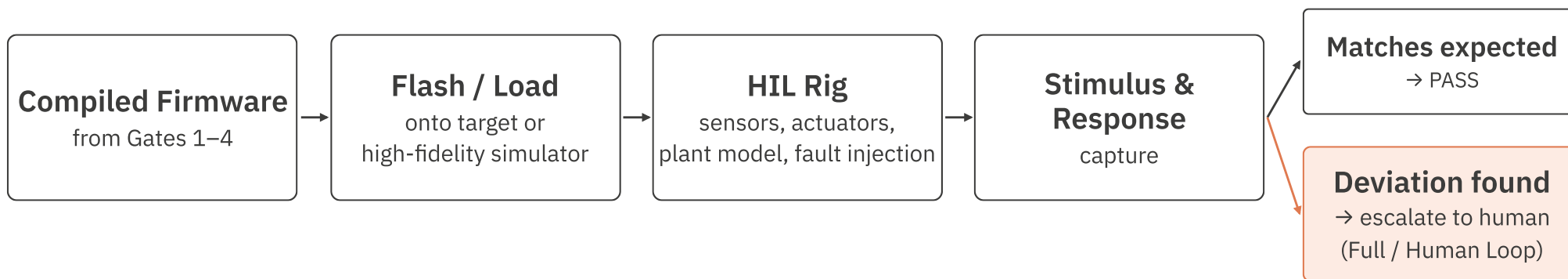


IAR solution: C-RUN: runtime bounds, arithmetic and heap-error detection with call-stack context. iar.com/embedded-development-tools/iar-c-run

Gate 5: Hardware-in-the-Loop

On-target integration testing

- The last gate before sign-off: the compiled binary runs on real (or high-fidelity simulated) hardware, not just in an analyser or a host build.
- A rig emulates the sensors, actuators and plant the firmware would see in the field, and drives it with realistic and fault-injected signals.
- The slowest, human-supervised loop — a failure here is an escalation, not an instant automated retry.



(Limited to UI testing) — Squish's host/target automation loop, separate from the HIL rig above:



IAR solution: C-SPY debug the device during HIL tests, hot-swap to connect/reconnect the probe mid-session. [iar.com/embedded-development-tools/iar-embedded-workbench](https://www.iar.com/embedded-development-tools/iar-embedded-workbench)

Qt Group solution: Squish (limited to UI testing): UI test automation over a host/target link — see diagram above. qt.io/quality-assurance/squish

Agent is squishserver, or Device Link on MCU targets; commands travel over network (or serial, target-dependent).

Act V:

the safety case

Gate qualification under ISO 26262-8:2018 (clause 11.4.5)

- Any tool used to develop safety-critical electronics has to be assessed:
→ if it malfunctions, that failure needs to be caught before it can violate a safety requirement.
- ISO 26262 structures that assessment around three terms:

■ **Tool Impact (TI)** ■ **Tool error Detection (TD)** ■ **Tool Confidence Level (TCL)**

The classification of a tool starts by first deciding on TI and TD for the tool (gate) under analysis.

TI — Tool Impact

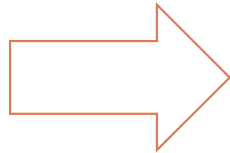
Evaluation of the potential of a tool to introduce or fail to detect errors

- **TI1**: there is an argument that there is no such possibility.
- **TI2**: the default for any code generator since it can introduce errors.

TD — Tool error Detection

Confidence that a malfunction will be prevented or detected:

- **TD1**: High confidence.
- **TD2**: Medium confidence.
- **TD3**: All other cases



The setting of TI and TD ultimately determines the TCL

TCL — Tool Confidence Level

$$\text{TCL} = \text{TI} \times \text{TD}$$

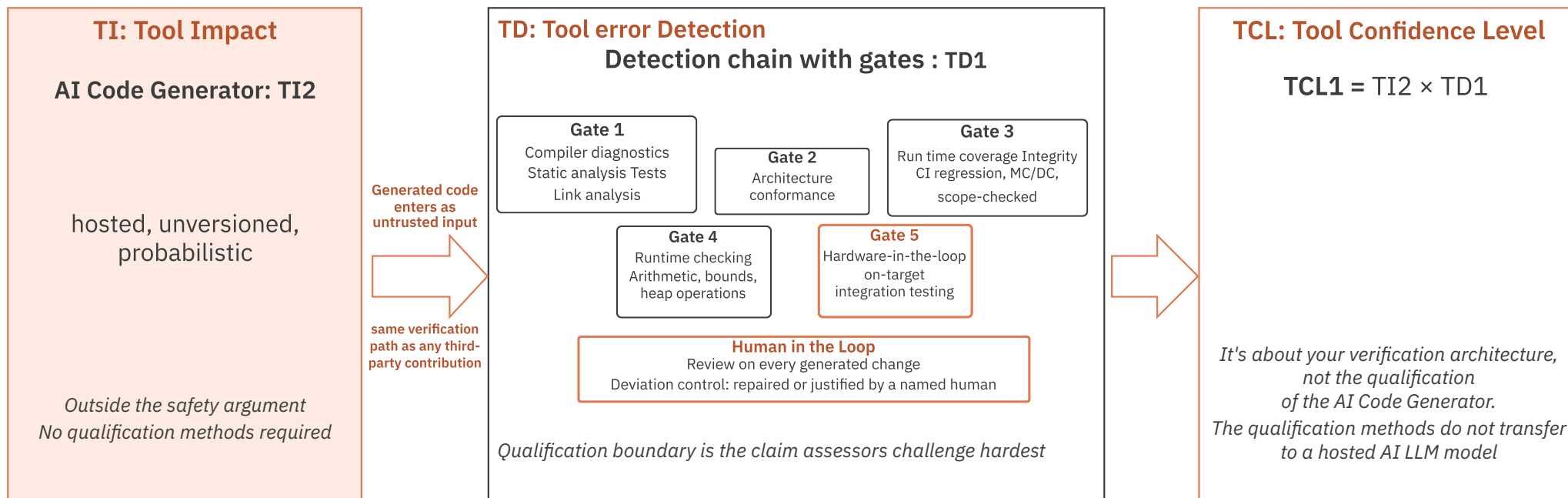
- **TCL1**: High level of confidence. No further qualification methods are required.
- **TCL2/TCL3**: Medium/Low level of confidence. Further qualification methods are required.

		Tool error detection		
		TD1	TD2	TD3
Tool impact	TI1	TCL1	TCL1	TCL1
	TI2	TCL1	TCL2	TCL3

ISO 26262-8:2018, clause 11.4.5 , Clearing the Fog of ISO 26262 Tool Qualification
<https://blogs.sw.siemens.com/verificationhorizons/2022/04/13/clearing-the-fog-of-iso-26262-tool-qualification/>.

Qualify the gate, not the generator

- An AI code generator can introduce an error
 - We classify **the AI code generator TI2** and make no claim that it cannot introduce errors.
 - We claim **TD1 on a documented 6 mechanism detection chain** (5 Deterministic Gates plus Human in the Loop)



ISO 26262-8:2018, clause 11.4.5 , Clearing the Fog of ISO 26262 Tool Qualification
<https://blogs.sw.siemens.com/verificationhorizons/2022/04/13/clearing-the-fog-of-iso-26262-tool-qualification/>.

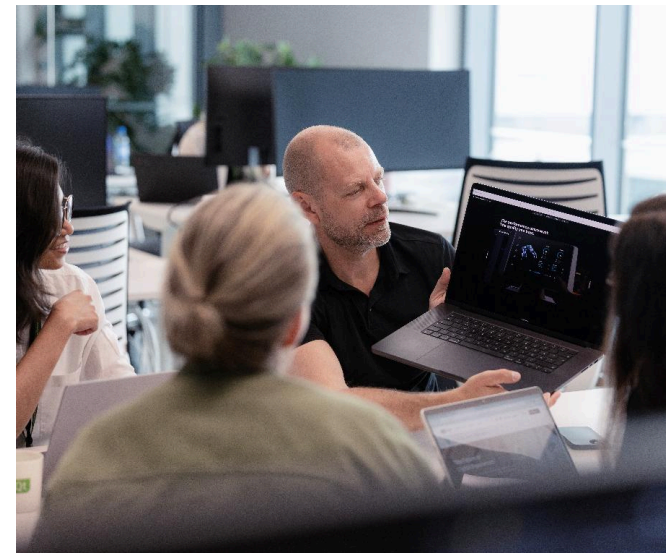
07

Conclusion:

Monday morning

Practices to adopt to make code compliant and safe

- Treat generated code as third-party code of unknown provenance
- Repeated runs across model tiers and effort levels
- The gate is what gets audited, not the tool defaults.
- Run deterministic analysis on every change in CI
- Use the model only for repair, not detection
- Human In the Loop : Require a written justification for every suppression



**Safety is a by-product of process, not of model capability.
In safety-critical systems, verification is the only form of trust on offer.**



—
**AI generates code.
It does not generate compliance.**



Thank you

Dr. Marc Thomas · Field Application Engineer

IAR Systems (Qt Group)

WoTS 2026 · Jaarbeurs Utrecht · 22 September 2026



CLOSING

REFERENCES

References

Bare URLs; validate every figure against the primary source before external use

- M. M. Umer, “Comparative Analysis of the Code Generated by Popular Large Language Models (LLMs) for MISRA C++ Compliance,” IEEE Access, vol. 13, pp. 194815–194831, 2025 (CC BY 4.0). <https://doi.org/10.1109/ACCESS.2025.3633086> · arXiv: <https://arxiv.org/abs/2506.23535>
- Spracklen, Wijewickrama, Sakib, Maiti, Viswanath, Jadhwal, “We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs,” Proc. 34th USENIX Security Symposium, 2025, pp. 3687–3706. <https://www.usenix.org/conference/usenixsecurity25/presentation/spracklen>
- N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, P. Liang, “Lost in the Middle: How Language Models Use Long Contexts,” Transactions of the ACL, vol. 12, pp. 157–173, 2024.
- Stack Overflow, 2025 Developer Survey — AI section (49,000+ respondents, 177 countries). <https://survey.stackoverflow.co/2025/ai/>
- Black Duck, The State of Embedded Software Quality and Safety 2025 (Censuswide, n = 785 embedded professionals, June 2025). <https://news.blackduck.com/2025-08-26-New-Black-Duck-Research-Shows-AI-and-Supply-Chain-Transparency-Redefining-Embedded-Software-Landscape>
- Sonar, State of Code Developer Survey, 8 January 2026 (n = 1,100+). Reported. <https://www.sonarsource.com/blog/ai-coding-trust-gap>
- Veracode, “2025 GenAI Code Security Report” (Jul 2025). https://www.veracode.com/wp-content/uploads/2025_GenAI_Code_Security_Report_Final.pdf · Oct 2025 Update (bundles original): <https://www.veracode.com/resources/analyst-reports/2025-genai-code-security-report/> · ⚠ “Spring 2026” report not found; verify before citing.
- Regulation (EU) 2024/2847 (Cyber Resilience Act). <https://digital-strategy.ec.europa.eu/en/policies/cyber-resilience-act>
- MISRA C:2025 position on AI-generated code — corroborated via secondary sources (Parasoft press release, Mar 2025; eeNews Europe; autosar.io).
- ISO 26262-8:2018, clause 11 (TI / TD / TCL) — mechanics Reported via consistent secondary sources; verify clause text against the licensed standard before quoting.
- “Reducing MISRA violations in LLM-generated code,” Research Square preprint, 25 November 2025. <https://doi.org/10.21203/rs.3.rs-8123173/v1>
- AWS Security Bulletin AWS-2025-015; CVE-2025-8217 (CWE-506); GHSA-7g7f-ff96-5gcw
- Siemens (Mentor) Verification Horizons, “Clearing the Fog of ISO 26262 Tool Qualification,” 13 April 2022 — corroborates Clause 11.4.5 as the location of the TI×TD→TCL determination table. <https://blogs.sw.siemens.com/verificationhorizons/2022/04/13/clearing-the-fog-of-iso-26262-tool-qualification/>
- M. Lawford, “Tools, Evidence, Arguments & Standards” (presentation slides), sos-vo.org — corroborates Clause 11.4.6 for the qualification methods; McMaster affiliation as stated in the source, not independently verified. https://www.sos-vo.org/system/files/sos_files/Tools_Evidence_Arguments_and_Standards.pdf



BACKUP

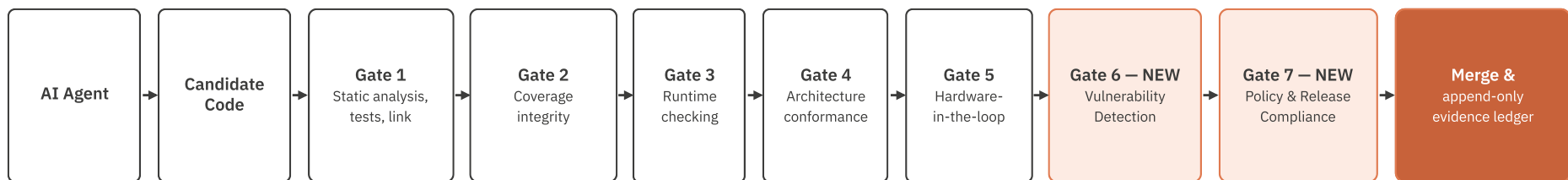
HIDDEN SLIDES

Annex Act VI: **the security case**

The fused deterministic gates — extended with security

- The five correctness gates from the main talk are unchanged — nothing here alters what Gates 1 to 5 check.
- Two more deterministic gates fold in the security side of the Cyber Resilience Act: vulnerability detection and policy-as-code enforcement.
- Same rule as before: FAIL returns a located finding, never a vague retry — the security gates behave no differently from the original five.

The 7-gate architecture required to make agentic code stand up in a safety case, security included:

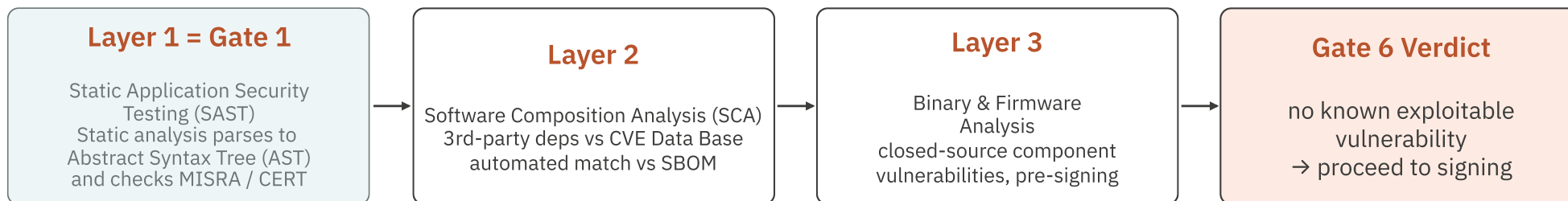


Gates 6–7 are new in this companion deck, grounded in the DevSecOps Pillars 3 & 4 work — not part of the main WoTS talk.

Every gate above — old or new — follows the same contract: FAIL returns a located finding to the agent; only a human resolves a REGRESSED or architecture-level result.

Gate 6 (Embedded Security): Vulnerability Detection

- Addresses the CRA's “no known exploitable vulnerabilities” requirement through multi-layered, automated scanning.
- Runs throughout the CI/CD workflow, not just as an end-stage check before release.
- Layer 1 (SAST) is exactly Gate 1 from the main talk, this gate's real addition is Layers 2 and 3: dependencies and binaries.



Same discipline as Gates 1–5: automated on every merge request, with a located finding on FAIL — never a silent block.

Result: code-quality validation time reduced from weeks to hours in the source case study.

Layer 1 (SAST) tools

Same mechanism as Gate 1 in the main talk — see Gate 1 for the full tool landscape (C-STAT, Parasoft, LDRA, Klocwork, Coverity, Polyspace, Cppcheck, Clang-Tidy, Frama-C). Not repeated here.

Layers 2–3 — SBOM / SCA ecosystem

STMicroelectronics, NXP, Wind River and Synopsys Black Duck are named as providing SBOM generation and vulnerability-verification support — genuinely new ground Gate 1 doesn't cover.

Gate 7 (Embedded Security): Policy & Release Compliance

- Automatically enforces CRA compliance rules using Open Policy Agent (OPA) — policies become executable code, not a checklist.
- Policies function as automated gates directly inside the CI/CD pipeline.
- This is the release gate: nothing reaches firmware signing without passing it.

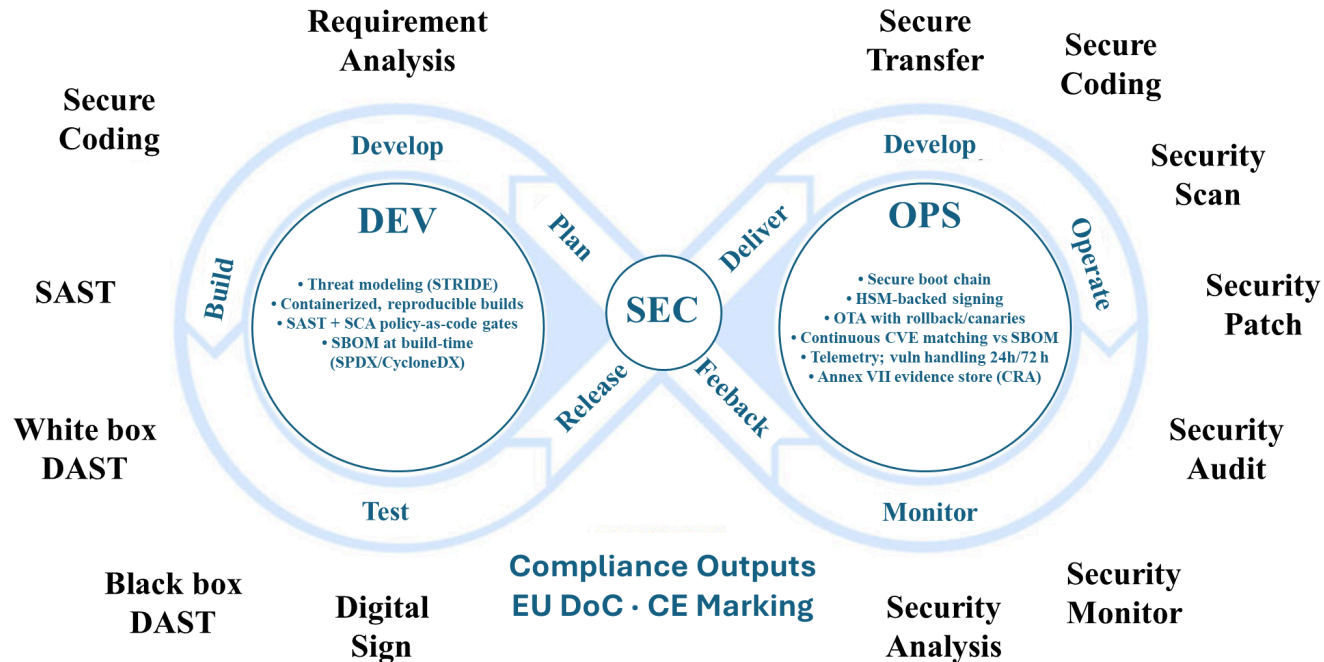


Example policy gates, not an exhaustive list; OPA policies are written per project.

Result: Automated, auditable evidence generation directly fulfilling the CRA's Annex VII compliance documentation requirements.

The DevSecOps automated loop: also valid for AI-generated code

- Security practices run continuously through the whole lifecycle, not bolted on at the end.
- The loop does not check who — or what — wrote the change. A human commit and an agent's commit enter it identically.



Gates 1–7 decide whether a change is correct and secure; this loop is where that decision keeps running, continuously, in production.

Annex Act II Backup :

the evidence under the microscope

Slopsquatting: package shared names are the attack surface

- **One continuous chain:** a plausible name, a repeatable one, a registered one, a resolved dependency
→ the SBOM that records the result as legitimate, since it attests to the attacker's package as faithfully as to any other

Shared names are the attack surface:

pip install aws-cdk is one of them, when the real package is **aws-cdk-lib**.

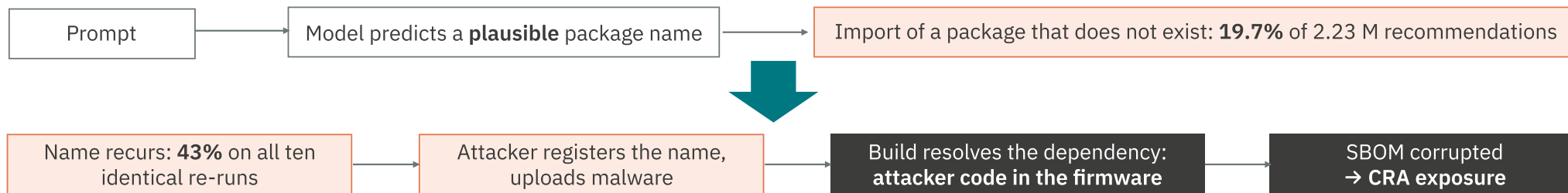
→ **Registration costs nothing: one sweep yields over 1300 hallucinated mentions per model.**

Spracklen et al., "We Have a Package for You!" Proc. 34th USENIX Security Symposium, 2025, pp. 3687–3706.

Churilov, "The Range Shrinks, the Threat Remains", arXiv:2605.17062 (v3, 9 August 2026) — preprint, not peer-reviewed; artifact: Zenodo 10.5281/zenodo.19859120.

47

Slopsquatting: from prompt to corrupted SBOM



Churilov (2026), a replication of the Spracklen study on frontier-tier models shows

127 package names (109 on PyPI, 18 on npm) invented identically by all five 2026 models
Claude Sonnet 4.6, Claude Haiku 4.5, GPT-5.4-mini, Gemini 2.5 Pro, DeepSeek V3.2



after PyPI and npm defences

53 still registrable
41 PyPI · 12 npm

reviewed by PyPI Security and Socket.dev, April 2026

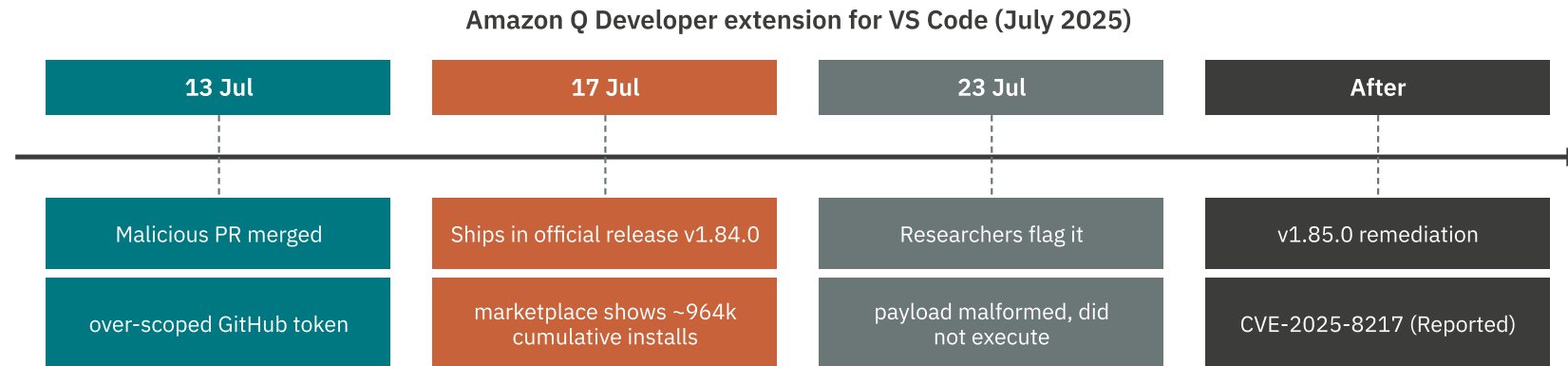
Spracklen et al., “We Have a Package for You!” Proc. 34th USENIX Security Symposium, 2025, pp. 3687–3706.

Churilov, “The Range Shrinks, the Threat Remains”, arXiv:2605.17062 (v3, 9 August 2026) — preprint, not peer-reviewed; artifact: Zenodo 10.5281/zenodo.19859120.

48

Payload stopped by a syntax error

- **Payload stopped by a syntax error:** July 2025, a wiper prompt shipped in Amazon Q Developer for VS Code v1.84.0
 - stopped by a malformed payload, not by a deterministic gate.

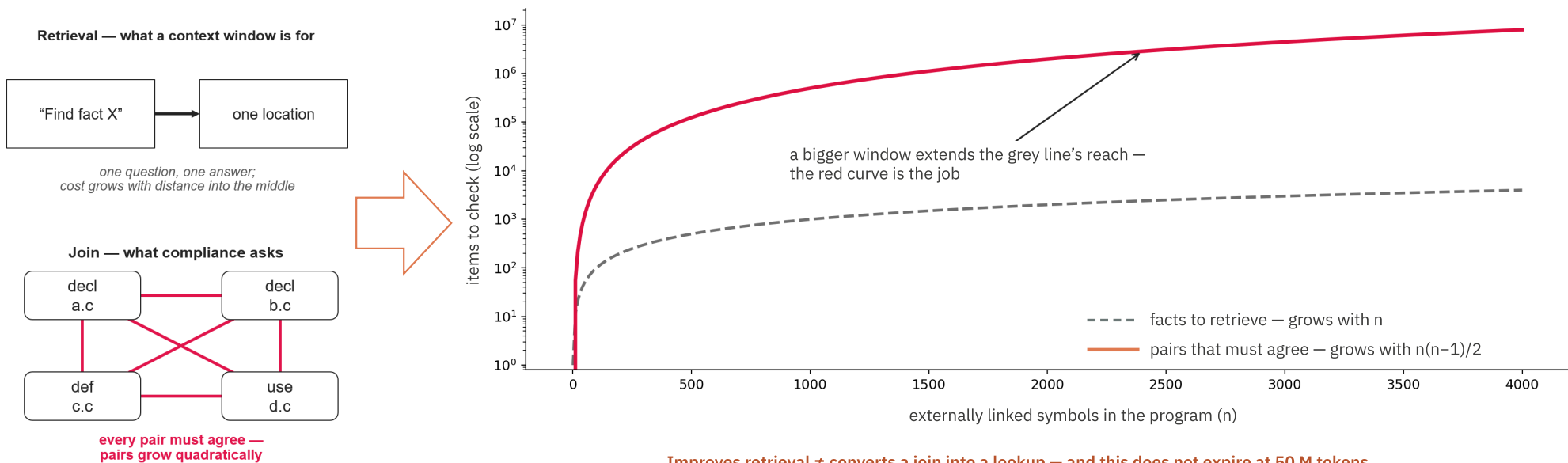


AWS Security Bulletin AWS-2025-015; CVE-2025-8217 — Reported: verify identifiers; 964k = cumulative installs, not exposure.

Compliance is a join, not a retrieval : why a bigger window does not solve it

- Window sizing ignores the preprocessor multiplier — vendor headers re-expand into every translation unit.
- Recall degrades in the middle of long inputs, and a program has no privileged position (“lost in the middle”).
- Compliance is an n-way consistency check over the symbol table:

→ **pairs grow quadratically, and an agent’s local scope hides exactly those collisions.**



Improves retrieval ≠ converts a join into a lookup — and this does not expire at 50 M tokens.

Token estimates ≈ 10 – 13 tokens per line

N. F. Liu et al., “Lost in the Middle,” Transactions of the ACL, vol. 12, pp. 157–173, 2024. 🔍 Verify against the publisher record.

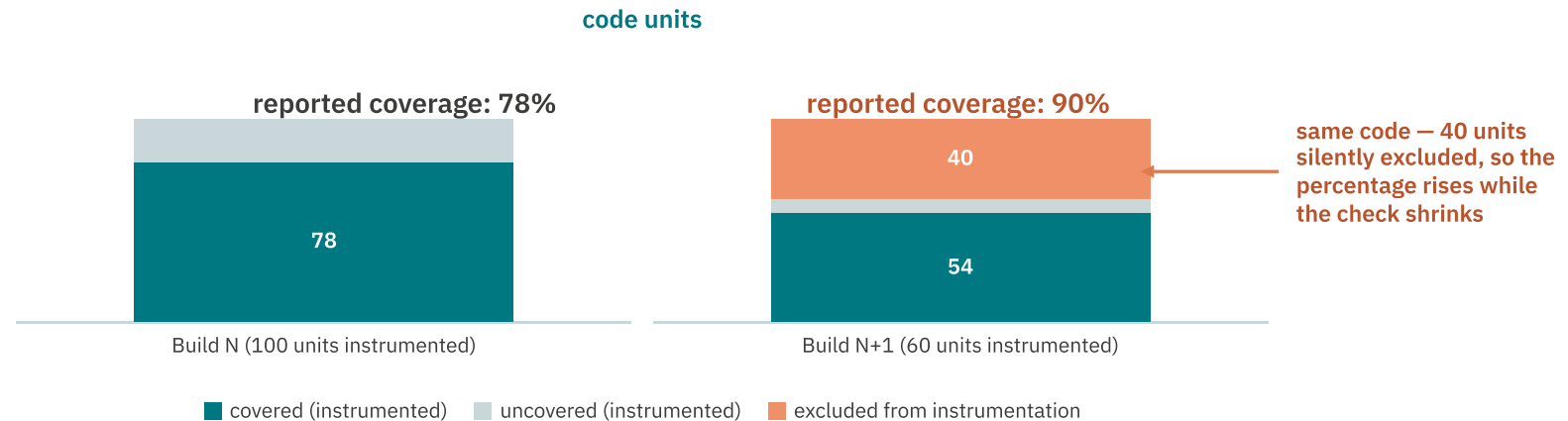
Generative model (probabilistic) vs Static analyser (deterministic)

- A system that generates without an oracle is an **unbounded search with no acceptance test**:
→the analyser is the loop’s oracle and stopping condition.

	Generative model (probabilistic)	Static analyzer (deterministic)
Output	Varies run to run — a sample	Fixed logic — an answer
Core engine	Text prediction over tokens	Compiled abstractions / symbol tables
Codebase view	Fragmented context window	Whole-program translation units
Role	Generate & repair (local)	Detect & locate (global)

Gate 2 (Code Coverage) blind spot: the silent denominator

- Coverage counts instrumented code only; architecture checks cover mapped code only:
→ exclusions shrink numerator and denominator invisibly.
- An agent's cheapest lever is shrinking the denominator, not improving the numerator.
- Rule: every percentage gate needs a companion check on its population — a scope-integrity monitor (a proposal, not a shipped feature).



Illustrative numbers. The mechanism is documented: coverage counts instrumented code only; architecture checks cover mapped code only.

Documented tool behaviour, joined across the gate-3 tools (series) — the scope-integrity monitor is a proposal, not a shipped feature.

CLOSING

Glossary & Abbreviations

Annex A — Glossary (Concise) — 1/3

One line each. Full definitions and Act references are in Annex B.

Absence — Gate 4 verdict: a dependency the architecture model expects but the code doesn't have. No code location, so never auto-repaired.

AI — Artificial intelligence.

Anchor study — Umer, IEEE Access 2025 — five LLMs, one MISRA C++ instruction, zero fully compliant.

ASLR — Address Space Layout Randomization.

AST — Abstract syntax tree — the parsed structure a static analyser reasons over.

CC BY 4.0 — Creative Commons Attribution 4.0 International licence.

CE marking — Conformité Européenne — the EU conformity mark a CRA-covered product carries at release.

CERT (coding standard) — Secure-coding rule set from the CERT Division, Carnegie Mellon Software Engineering Institute.

CI / CI/CD — Continuous integration / continuous integration and delivery.

Convergence — Gate 4 verdict: the code's actual dependency graph matches the architecture model.

CRA — Cyber Resilience Act — Regulation (EU) 2024/2847.

CVE — Common Vulnerabilities and Exposures — a public vulnerability identifier.

CWE — Common Weakness Enumeration — a vulnerability *class* identifier (contrast CVE, a specific instance).

Deterministic gate — A check that returns the same verdict on the same input, every time — the property the thesis rests on.

Divergence — Gate 4 verdict: a forbidden dependency the code has but the architecture model doesn't allow. Has a code location.

DOI — Digital Object Identifier.

ECU — Electronic control unit.

Enforced (check) set — The versioned, specific list of rules a gate actually treats as blocking — what an audit checks against, not the tool's shipped defaults.

Five-gate architecture — This talk's core proposal: static analysis, coverage integrity, runtime checking, architecture conformance, hardware-in-the-loop.

Gate outcome states — PASS / FAIL / REGRESSED / INCONCLUSIVE — see individual entries.

GHSA — GitHub Security Advisory identifier.

GPT — Generative Pre-trained Transformer — a model architecture family (GPT-4o, GPT-5, etc.).

HIL — Hardware-in-the-loop — Gate 5.

IEEE — Institute of Electrical and Electronics Engineers.

INF Treaty — Intermediate-Range Nuclear Forces Treaty, signed 8 December 1987 — source of "trust, but verify."

ISO 26262 — Road-vehicle functional-safety standard; Part 8 Clause 11 governs tool qualification.

IWYU — Include-What-You-Use — a header-dependency hygiene tool.

KLOC / MLOC — Thousand / million lines of code.

LLM — Large language model.

Located finding — A gate's output when it fails: file, line, rule — never a vague warning.

Annex A — Glossary (Concise) — 2/3

One line each. Full definitions and Act references are in Annex B.

Lost in the middle — Long-context LLM recall is weakest for material in the middle of the input, regardless of file order.

MC/DC — Modified Condition/Decision Coverage — the coverage criterion required at the highest safety levels.

MCU / MPU — Microcontroller unit / microprocessor unit.

MISRA — Motor Industry Software Reliability Association; publisher of the MISRA C/C++ coding guidelines.

npm — Node.js package manager. Δ npm's own documentation states it is not an acronym; "Node Package Manager" is the common gloss, not an official expansion — verify before printing an expansion as fact.

Oracle — In this talk, the thing whose verdict is trusted as ground truth. The deterministic analyser is the oracle; the LLM cannot be, because its verdict varies run to run.

ODR — One Definition Rule — the C++ rule that a link-time analysis pass checks for violations of.

OPA — Open Policy Agent — policy-as-code engine used for Gate 7.

PASS / FAIL / REGRESSED / INCONCLUSIVE — The deck's four machine-readable gate outcomes. FAIL = a code problem, located; REGRESSED = a previously-passing property broke; INCONCLUSIVE = the stage couldn't run at all (never treat as PASS).

PSF — Python Software Foundation.

PyPI — Python Package Index.

Reagan paradox — This talk's framing device: a Cold War verification protocol applied to AI-generated code.

RTOS — Real-time operating system.

Safety case — The documented argument, with evidence, that a system is acceptably safe — what Act V builds toward.

SAST — Static Application Security Testing — Gate 6, Layer 1 (same mechanism as Gate 1).

SARIF — Static Analysis Results Interchange Format.

SBOM — Software Bill of Materials.

SCA — Software Composition Analysis — Gate 6, Layer 2.

Scope-integrity monitor — This project's own proposed safeguard against a silently shrinking coverage/architecture denominator. Not a shipped feature of Coco or Axivion.

Shadow AI — Use of AI coding tools outside sanctioned policy.

Silent denominator — The population a percentage is computed over, which can shrink invisibly (fewer files instrumented, fewer dependencies mapped) while the percentage itself rises.

Slopsquatting — Registering a package name an LLM reliably hallucinates, so the next developer who follows the suggestion installs the attacker's code.

SQL — Structured Query Language.

Symbol table — A compiler's authoritative record of every declared name — what an analyser has and a context window doesn't.

TCL — Tool Confidence Level (ISO 26262-8) — the combination of TI and TD.

Annex A — Glossary (Concise) — 3/3

One line each. Full definitions and Act references are in Annex B.

TD — Tool error Detection (ISO 26262-8) — confidence a tool malfunction would be caught.

TI — Tool Impact (ISO 26262-8) — whether a tool malfunction can introduce or miss an error.

Translation unit — A source file plus everything the preprocessor pulls into it — the real scope of many compliance rules.

USENIX — The Advanced Computing Systems Association, historically the Unix Users Group; publisher of the slopsquatting study's venue (USENIX Security).

VS Code — Visual Studio Code.

Wiper — Malware whose purpose is destruction rather than theft or persistence.

XSS — Cross-site scripting — CWE-80.

Annex B — Glossary (Detailed) — 1/8

- **Absence** — An architecture-conformance verdict (Gate 4): a dependency the architecture model requires that does not occur in the actual code. Unlike a Divergence, an Absence has no code location to point to, so — like a Divergence — it is never auto-repaired; it goes to human triage. *Act IV, Gate 4.*
- **Agent / Agentic tool** — An AI assistant that acts on a codebase rather than only suggesting text: it edits files, runs the build, reads diagnostics, and edits again. The deck's framing is that this changes the *rate* at which code arrives at the gate, not what the gate must check. *Introduction; throughout.*
- **AI** — Artificial intelligence; used throughout without further definition.
- **ASLR** — Address Space Layout Randomization. Appears as an example build-configuration check enforced at Gate 7 (policy: fail the build if stack protection or ASLR settings are missing). *Act IV, Gate 7.*
- **AST** — Abstract syntax tree. The parsed representation a static analyser checks against a coding-standard rule set (Gate 1), and what SAST (Gate 6, Layer 1) parses to as well. *Act IV, Gates 1 and 6.*
- **CC BY 4.0** — Creative Commons Attribution 4.0 International licence, under which the anchor study (Umer, IEEE Access) is published open access. *Act II.*
- **CE marking** — Conformité Européenne. Listed on the CRA timeline as one of the deliverables tied to the essential-requirements deadline (11 December 2027). *Act I.*
- **CERT (coding standard)** — Secure-coding rule set from the CERT Division of Carnegie Mellon's Software Engineering Institute, named alongside MISRA as a rule set Gate 1 and Gate 6/SAST check against. The deck does not expand the acronym on-slide. *Act IV, Gates 1 and 6.*
- **CI / CI/CD** — Continuous integration / continuous integration and continuous delivery. The deck's Monday-morning practice list calls for running analysis "on every change in CI, with pinned determinism and explicit timeouts"; Gates 6–7 are described as running "throughout the CI/CD workflow," not only as an end-stage check. *Conclusion; Act IV backup (Gates 6–7).*

Annex B — Glossary (Detailed) — 2/8

- **Convergence** — Gate 4 verdict: the code's actual, statically-extracted dependency graph matches the intended architecture model. The "OK" case, contrasted with Divergence and Absence. *Act IV, Gate 4.*
- **CRA** — Cyber Resilience Act, Regulation (EU) 2024/2847. Three dates matter and the deck keeps them distinct rather than telescoped: entry into force December 2024, Article 14 incident-reporting duty from 11 September 2026 (24h early warning, 72h notification, 14-day final report), essential requirements and CE marking from 11 December 2027. The deck's argument: Article 13's "no known exploitable vulnerabilities" standard doesn't ask who or what wrote the code, which is the premise the whole talk is built on. *Act I.*
- **CVE** — Common Vulnerabilities and Exposures. A specific, publicly-tracked vulnerability instance (contrast CWE, a *class*). CVE-2025-8217 is cited (△ Reported, verify identifier) against the Amazon Q incident. *Act I.*
- **CWE** — Common Weakness Enumeration. A vulnerability *class* taxonomy. CWE-506 (Amazon Q incident), and CWE-80/89/117/327 (cross-site scripting, SQL injection, log injection, weak cryptography — the Veracode CWE breakdown backup slide) both appear. *Act I; backup.*
- **Deterministic gate** — A check whose verdict does not vary run to run on identical input: the property that lets it carry compliance and regulatory weight, contrasted throughout with the LLM's probabilistic output. *Introduction; Act III; Act IV.*
- **Deviation (management)** — The human-owned practice of recording, justifying, and approving an accepted exception to a rule, rather than silencing the finding. The deck's Monday-morning list requires the justifier not be the AI author. *Conclusion.*
- **Divergence** — Gate 4 verdict: a forbidden dependency present in the code's actual structure that the architecture model does not allow. Has a code location, but is still never auto-repaired, because the deck's own caution is that the cause could be the code, the mapping, *or* the architecture model itself being stale — and the agent cannot tell which. *Act IV, Gate 4.*
- **DOI** — Digital Object Identifier. Used throughout the References slide's citations.
- **ECU** — Electronic control unit. Used as the top tier of the deck's firmware-scale table (0.5–2 MLOC, 5–26 M source tokens) to argue no context window covers it. *Act III.*
- **Enforced (check) set** — The specific, versioned list of rules a gate actually treats as blocking, distinct from whatever a tool ships enabled by default. The deck's Monday-morning practice list frames this as what gets audited. *Conclusion.*

Annex B — Glossary (Detailed) — 3/8

- **GHSA** — GitHub Security Advisory identifier (e.g. GHSA-7g7f-ff96-5gcw, cited against the Amazon Q incident). *Act I.*
- **GPT** — Generative Pre-trained Transformer, the model architecture family behind the GPT-4o mini and GPT-5-series models named in the deck (anchor study, Veracode October 2025 update discussion).
- **HIL (Hardware-in-the-Loop)** — Gate 5, the last and slowest gate: the compiled binary runs on real or high-fidelity simulated hardware, driven by a rig emulating sensors, actuators, and plant behaviour with fault injection. A failure here escalates to a human rather than triggering an automated retry. *Act IV, Gate 5.*
- **IEEE** — Institute of Electrical and Electronics Engineers; publisher (via IEEE Access) of the anchor study. *Act II; References.*
- **INF Treaty** — Intermediate-Range Nuclear Forces Treaty, signed 8 December 1987, where Reagan quoted the Russian proverb "doveryai, no proveryai" ("trust, but verify"). The deck's framing device: the treaty held because of its inspection machinery, not interpersonal trust — the same move it proposes for AI-generated code. △ The deck is careful to attribute the quotation and the treaty context with confidence, while noting (per this project's own established caveat) that confidence about substance doesn't license confidence about the exact wording of any one delivery of the line. *Introduction.*
- **ISO 26262** — Road-vehicle functional-safety standard. Part 8, Clause 11 (specifically 11.4.5) governs software tool qualification and is the basis for Act V's TI/TD/TCL argument. ● Clause mechanics are reported via consistent secondary sources (a Siemens Verification Horizons post is the deck's cited corroboration);
- **IWYU (Include-What-You-Use)** — A header-dependency hygiene tool, named in the Gate 4 tool-landscape table as covering a narrower slice of the architecture-conformance problem. *Act IV, Gate 4.*
- **KLOC / MLOC** — Thousand / million lines of code. Used in the firmware-scale table (Act III) and in the anchor-study discussion, where the deck notes per-model counts are absolute, not normalised by file size — so verbosity and quality are confounded in the raw figures.
- **LLM** — Large language model: "a probabilistic, token-based generator with no internal static-analysis mechanism against which to validate its own output" (the deck's own working definition, Act III). Specific models named across the deck: ChatGPT/GPT-4o mini, DeepSeek V3 and V3.2, Gemini 2.5 Flash and Pro, Meta AI/LLaMA 4, Microsoft Copilot, Claude Sonnet 4.6, Claude Haiku 4.5, GPT-5.4-mini.
- **Located finding** — A gate's failure output: rule ID, severity, file, and line — "never a vague warning" is the deck's own repeated phrase. The contract every gate (1 through 7, where the 7-gate extension applies) is described as sharing. *Act IV.*
- **Lost in the middle** — Liu et al., *Transactions of the ACL*, 2024: long-context LLM recall is strongest at the edges of an input and weakest in the middle, regardless of where in a file the relevant material actually sits — "the program has no privileged position." Used to argue that a bigger context window doesn't fix compliance checking, because compliance is an n-way consistency check (a *join*), not a single-fact lookup (a *retrieval*), and join costs grow quadratically with the number of linked symbols. *Act III.* ● One citation of this paper on a backup slide is marked "verify against the publisher record."
- **MC/DC (Modified Condition/Decision Coverage)** — The coverage criterion the deck describes as "auditor-grade," required at the highest safety levels: every condition must be shown to independently affect the decision outcome. Gate 2's output. *Act IV, Gate 2.*
- **MCU / MPU** — Microcontroller unit / microprocessor unit. Used in the firmware-scale table's size classes (sensor node, connected MCU+RTOS, rich MCU/MPU application, automotive ECU). *Act III.*

Annex B — Glossary (Detailed) — 4/8

- **MISRA** — Motor Industry Software Reliability Association, publisher of the MISRA C/C++ coding guidelines that are the rule set behind Gate 1 and the anchor study's compliance target. The deck cites "MISRA C:2025: code origin is irrelevant, the same guidelines apply to AI-generated code"
- **npm** — Node.js package manager, the JavaScript-ecosystem counterpart to PyPI in the slopsquatting discussion (127 shared hallucinated names span both PyPI and npm). Δ npm's own documentation states the name is not an acronym; "Node Package Manager" is a common gloss, not an official expansion. Verify before presenting an expansion as fact in the article. *Act I.*
- **ODR (One Definition Rule)** — The C++ rule that a link-time static-analysis pass (Gate 1, Phase 2) checks for violations of, alongside identifier distinctness and indirect recursion — issues no single-file analysis pass can see. *Act IV, Gate 1.*
- **OPA (Open Policy Agent)** — The policy-as-code engine behind Gate 7 (Policy & Release Compliance): CRA compliance rules become executable policy rather than a checklist, gating firmware signing. *Act IV backup, Gate 7.*
- **Oracle** — In this talk's own vocabulary: the thing whose verdict is trusted as ground truth for a check. The deterministic static analyser is the oracle because "independence and repeatability are the whole value of an oracle"; the LLM cannot be the oracle because its verdict "varies run to run: a sample, not an answer." The LLM can sit upstream (triage) or downstream (drafting deviation rationales) of the oracle, but never replace it. *Act III.*
- **PSF (Python Software Foundation)** — Credited (in the deck's own speaker notes, with a verify-before-saying flag) as Seth Larson's affiliation at the time slopsquatting was coined, April 2025. Δ Neither of the two academic sources cited on the slopsquatting slides credits Larson directly; the deck's own notes instruct verifying the attribution before saying it from the stage. *Act I.*
- **PyPI (Python Package Index)** — The Python-ecosystem package registry central to the slopsquatting mechanism: a model hallucinates a package name, an attacker registers it on PyPI (or npm), the build resolves the dependency, and the firmware ships attacker code the SBOM records as legitimate. *Act I.*
- **Reagan paradox** — This deck's own framing device, introduced in its title section: applying a Cold War arms-control verification principle to AI-generated code. Not a term with existence outside this talk. *Introduction.*
- **RTOS (Real-time operating system)** — Named in the firmware-scale table's second size class ("Connected MCU + RTOS"). *Act III.*
- **Safety case** — The documented, evidence-backed argument that a system is acceptably safe for its intended use — what Act V is building toward via the ISO 26262-8 TI/TD/TCL classification. *Act V.*
- **SAST (Static Application Security Testing)** — Gate 6, Layer 1 in the extended 7-gate architecture; explicitly described as "exactly Gate 1 from the main talk" — parses to AST, checks against MISRA/CERT. *Act IV backup, Gate 6.*
- **SARIF (Static Analysis Results Interchange Format)** — The interchange format Gate 1's located findings are described as being output in (rule ID, severity, file:line). *Act IV, Gate 1.*

Annex B — Glossary (Detailed) — 5/8

- **SBOM (Software Bill of Materials)** — The artefact that, in the slopsquatting chain, "attests to the attacker's package as faithfully as to any other" once a hallucinated dependency resolves at build time — the deck's stated reason this belongs in an embedded talk rather than only a security talk. Also the object Gate 6/Layer 2 (SCA) matches against a CVE database, and Gate 7 validates before release. *Act I; Act IV backup, Gates 6–7.*
- **SCA (Software Composition Analysis)** — Gate 6, Layer 2: matches third-party dependencies against a CVE database and the SBOM. Named partners: STMicroelectronics, NXP, Wind River, Synopsys Black Duck. *Act IV backup, Gate 6.*
- **Scope-integrity monitor** — This project's own proposed safeguard against the "silent denominator" problem (see below): a companion check on the *population* a coverage or architecture percentage is computed over, not just the percentage itself. The deck is explicit this is "a proposal, not a shipped feature" of Coco or Axivion. *Act IV backup (Gate 2 blind spot).*
- **Shadow AI** — Use of AI coding tools outside sanctioned organisational policy. The deck's cited figures: 18% of embedded organisations report it overall (Black Duck 2025); of the roughly one-in-four organisations that prohibit AI assistants outright, 75% still see developers using them anyway — "a ban removes visibility, not code." *Act I.*
- **Silent denominator** — The population a percentage gate is computed over (instrumented files for coverage; mapped code for architecture conformance), which can shrink invisibly — fewer units instrumented, fewer dependencies mapped — while the reported percentage itself rises. The deck's illustrative example: coverage reads 90% on a smaller, shrunk population where it read 78% on a larger one, with the same 40 units silently excluded. *Act IV backup (Gate 2 blind spot).*
- **Slopsquatting** — Registering a package name that a large language model reliably and repeatably hallucinates, so that the next developer who follows the model's suggestion installs the attacker's package instead. The deck's own two-source treatment: Spracklen et al. (USENIX Security 2025) measured 19.7% of 2.23M package recommendations as hallucinated, with 43% of hallucinated names recurring on all ten identical re-runs (though 39% never recur at all — persistence is bimodal, and the attacker only needs the stable half). Churilov (2026, Δ preprint, not peer-reviewed) replicated the study on five frontier models and found 127 hallucinated names shared identically across all five, of which 53 remain registrable after PyPI/npm's own defences. *Act I.*
- **SQL (Structured Query Language)** — Named via SQL injection (CWE-89), one of the four vulnerability classes in the Veracode CWE breakdown. *Backup.*
- **Symbol table** — A compiler's authoritative, de-duplicated record of every declared name across a program — the structural advantage an analyser has that a context window cannot replicate, because a context window pays token cost for every re-expansion of a header rather than collapsing it once. *Act III.*
- **TCL (Tool Confidence Level)** — ISO 26262-8's combination of Tool Impact (TI) and Tool error Detection (TD) into one of three confidence classes. TCL1 (the lowest, not a "top" award) requires no further qualification measures; TCL2/TCL3 do. The deck's own safety-case claim is $TI2 \times TD1 = TCL1$, built on an eight-mechanism detection chain rather than on any claim about the generator itself. *Act V.*
- **TD (Tool error Detection)** — ISO 26262-8's measure of confidence that a tool malfunction and its erroneous output would be prevented or detected — TD1 is high confidence, and the deck is explicit this is a claim about the *verification architecture*, not about the tool's own reliability, which is exactly why it's the lever the deck's argument pulls. *Act V.*
- **TI (Tool Impact)** — ISO 26262-8's measure of whether a tool malfunction can introduce or fail to detect an error in the safety-related item. The deck classifies any AI code generator as TI2 without argument — "attempting a TI1 claim damages credibility" — since a generator manifestly can introduce errors. *Act V.*

Annex B — Glossary (Detailed) — 6/8

- **Translation unit** — A source file plus everything the preprocessor pulls into it (headers, macros) — the actual scope a compiler and a per-file static-analysis pass see, and the unit the deck's firmware-scale argument is built on (vendor headers re-expand into every translation unit, multiplying the token cost of a context-window approach). *Act III; Act IV, Gate 1.*
- **USENIX** — Publisher, via its Security Symposium, of the Spracklen et al. slopsquatting study. $\triangle$ Historically associated with the Unix Users Group; the organisation's current preferred self-description is "The Advanced Computing Systems Association," and it does not consistently treat the name as a strict acronym — verify current usage before expanding it in the article. *Act I; References.*
- **Wiper** — Malware whose purpose is destruction rather than theft or persistence — the payload class involved in the Amazon Q incident. *Act I.*
- **XSS (Cross-site scripting)** — CWE-80, one of the four vulnerability classes in the Veracode CWE breakdown; the class Veracode's own report says LLMs perform worst on, because avoiding it requires interprocedural taint-tracking of which variables carry untrusted data — a computation the deck argues "it is unlikely LLMs will ever be able to perform... directly." *Backup.*

Annex C — List of Abbreviations

Expansions only. ⚠ = common usage, not independently confirmed — see Annex B.

AI	Artificial Intelligence	ECU	Electronic Control Unit	MCU	Microcontroller Unit	SBOM	Software Bill of Materials
ASLR	Address Space Layout Randomization	EU	European Union	MISRA	Motor Industry Software Reliability Association	SCA	Software Composition Analysis
AST	Abstract Syntax Tree	GHSA	GitHub Security Advisory	MLOC	Million Lines of Code	SQL	Structured Query Language
CC BY 4.0	Creative Commons Attribution 4.0 International	GPT	Generative Pre-trained Transformer	MPU	Microprocessor Unit	TCL	Tool Confidence Level
CE	Conformité Européenne	HIL	Hardware-in-the-Loop	npm	⚠ Node.js package manager (not an acronym)	TD	Tool error Detection
CERT	Carnegie Mellon SEI coding-standard division	IEEE	Institute of Electrical and Electronics Engineers	ODR	One Definition Rule	TI	Tool Impact
CI	Continuous Integration	INF (Treaty)	Intermediate-Range Nuclear Forces	OPA	Open Policy Agent	USENIX	⚠ Unix Users Group, now Computing Systems Assoc.
CI/CD	Continuous Integration / Continuous Delivery	ISO	International Organization for Standardization	PSF	Python Software Foundation	VS Code	Visual Studio Code
CRA	Cyber Resilience Act	IWYU	Include-What-You-Use	PyPI	Python Package Index	WoTS	⚠ Usually "World of Tech. and Science"
CVE	Common Vulnerabilities and Exposures	KLOC	Thousand Lines of Code	RTOS	Real-Time Operating System	XSS	Cross-Site Scripting
CWE	Common Weakness Enumeration	LLM	Large Language Model	SAST	Static Application Security Testing		
DOI	Digital Object Identifier	MC/DC	Modified Condition/Decision Coverage	SARIF	Static Analysis Results Interchange Format		