
LLM Trainer — Code Security

About Solumn AI

Solumn AI builds the safety and alignment layer for frontier AI systems.

We work with leading AI labs to make advanced models deployment-ready, across the post-training and assurance stack: red teaming, vulnerability discovery, safety evaluations, agent-behavior testing, and the datasets and frameworks that make it measurable. If capability asks "what can the model do?", we ask "what happens when it fails, and how do we prevent it?"

Why this matters: Every technology cycle begins with capability and matures through trust. The internet, cloud, mobile and social all ran the same arc, with security and governance becoming indispensable only after the first wave of scale. AI is crossing that threshold now, and the assurance layer is becoming foundational. We're building the firm that does it to a benchmark standard.

Role Overview

We are looking for an **LLM Trainer — Code Security** to review and evaluate how AI coding agents behave on two closely related surfaces: **software-engineering trajectories** (an agent reading, writing, debugging and patching real code) and **security trajectories** (an agent finding, analysing, exploiting or fixing real vulnerabilities).

This is a **high-judgment review role**. You will read what an agent actually did across a multi-turn run, decide whether the outcome was genuinely correct and genuinely secure, and write up the reasoning so a researcher can act on it directly. Your verdicts become training signal and safety evidence.

Agents routinely reach the right-looking answer for the wrong reason — patching a symptom, editing the test harness, overfitting to a proof-of-concept, or disabling the feature that carried the bug. Catching that requires someone who can analyse the vulnerability.

Key Responsibilities

Review software-engineering trajectories

- Read multi-turn agent runs on real repositories and assess whether the reasoning, tool use, and final patch were sound
- Evaluate the **diff itself**: does it fix the actual root cause, is it complete, does it introduce regressions or side effects
- Identify where an agent reached a passing result illegitimately — modifying tests or the harness, special-casing the checked input, or removing the failing behavior rather than fixing it
- Label failure modes precisely: incorrect reasoning, incomplete execution, silent scope creep, unsafe file or command operations

Review security trajectories

- Assess vulnerability-discovery runs — did the agent locate the true vulnerable code path, or a plausible-looking but incorrect one
- Judge fix quality against the vulnerability, not against the reference patch: **would a different, correctly-written fix also be accepted?** Flag verifiers and graders that only accept one implementation
- Check that a recorded pass is real rather than an artifact — proof-of-concept overfitting, a suppressed rather than resolved crash, or a fix applied outside the intended source
- Reason about common vulnerability classes (OWASP Top 10, CWE/SANS Top 25) well enough to tell a genuine remediation from a superficial one

Fix what you find

- Repair defective tasks rather than only rejecting them — correct an accept-set pointing at the wrong location, tighten one that admits spurious passes, or re-author a verifier coupled to a single implementation
- Write or amend security regression tests so they genuinely fail on the vulnerable build and pass on the fixed one
- Know the line between what you should repair yourself and what has to go back upstream — and never quietly edit the ground truth to make a task pass
- Turn a recurring defect into a rule others can apply, so the same class stops arriving

Deliver usable verdicts

- Produce consistent, well-evidenced evaluations that researchers can use directly for training data and safety assessment
- Cite the specific line, command, or episode that justifies the call — never a general impression
- Surface systematic problems in tasks, graders and rubrics, not just per-task verdicts

Required Experience

- **1–3 years** hands-on in one or more of:
 - Application or product security, VAPT, secure code review, or vulnerability research
 - Software engineering with real code-review responsibility
 - LLM evaluation, annotation, or red teaming for coding or tool-calling agents
 - (Strong security fundamentals and demonstrable hands-on work matter more here than years or pedigree.)
- **You can read code and judge a patch.** Comfortable in at least one of Python, JavaScript/TypeScript, Go, Java, C/C++, Ruby or PHP, and able to reason about a diff in a language you don't write daily
- **Working knowledge of vulnerability classes** — injection, path traversal, deserialization, memory safety, authentication and access-control flaws, and how each is properly fixed
- **Able to read long agent trajectories** — 10+ turn reasoning and tool-call chains — without losing the thread
- **Able to fix, not just flag.** You can write or repair a test that fails on the vulnerable build and passes on the fixed one, and correct a broken verifier or accept-set without weakening what it checks
- **Clear written reasoning.** Your annotations are the deliverable; they must stand up to a researcher disagreeing with them

- Comfortable with Git, containers/Docker, reading test output and stack traces, and structured labeling formats (JSON/JSONL)
-

Nice to Have

- CTF, bug bounty, or CVE credit
 - Experience writing or reviewing security regression tests, fuzzing harnesses, or exploit proofs-of-concept
 - Familiarity with SAST/DAST/SCA tooling and its false-positive behaviour
 - Exposure to agentic RL, reward modeling, or LLM-as-judge evaluation frameworks
 - Experience with MCP or tool-calling systems
 - Open-source security or evaluation contributions
-

What Success Looks Like

- Your verdicts are trusted with minimal re-review, and hold up when challenged
 - You reliably catch the passes that shouldn't have passed — the fix that games the test, the accept-set that only admits one answer, the crash that was masked rather than resolved
 - You are equally careful in the other direction: an agent that solves a problem differently from the reference is a pass, not a failure
 - Defective tasks come back from you repaired and usable, not just labelled — your review raises how much of a batch survives
 - You become the person researchers ask when a case is genuinely ambiguous
 - You spot the systematic defect behind a cluster of individual ones, fix the class rather than the instance, and say so
-

Role Type

- **Full-time / Part-time**
 - **High-judgment work** — calibrated reasoning on hard cases and hands-on repair, not checklist execution
 - Sits between hands-on security practice and AI evaluation; a strong on-ramp into frontier AI safety work for a security or engineering background
 - Remote
-