

Application Tested

Tester

AI-refactored Code



NCS, a pan APAC, Singapore-headquartered technology services firm, wanted to assess the performance of an in-house AI coding assistant, Sunshine.Coder, which leverages open-source code LLMs to refactor existing Java applications.




Parasoft specialises in automated software testing, focusing in DevSecOps solution for building security into critical programs like the US Department of Defence’s Joint Federated Assurance Center.

How LLMs were used in application?

Code generation (refactoring)

What Risks Were Considered Relevant And Tested?




Key question: Is AI-Refactored Code Safer and Smarter?

-  **Security, Compliance, and Standards Risks**
AI-refactored code may:
- Inadvertently introduce security vulnerabilities, such as improper input validation, insecure data handling, or failure to enforce authentication and authorisation controls
 - Not consistently adhere to internal corporate coding standards or external regulatory requirements (e.g., cybersecurity guidelines, data protection regulations)


→ Such deviations may cause technical debt, reduce auditability, and create hidden compliance risks

How Were The Risks Tested?

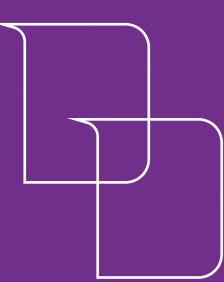
Approach

-  Testing conducted on two different codebase profiles – Production and Proof-of-Concept (PoC)
-  Issues mapped to compliance violations (e.g., deviations from security guidelines or coding standards)
-  Testing conducted without executing the applications, focusing purely on static structural analysis

Evaluators

-  **Rule based logic:**
Parasoft’s Jtest automated testing suite for code quality/security

Challenges



Variability in the quality of AI-refactored code across different applications and LLM configurations

Insights

01

Sunshine.Coder improves code quality, but starting point determines quality of AI refactored code

- Production-grade code inherently follows best practices, reducing the complexity needed in prompts
- PoC code, lacking best practice adherence, requires explicit prompts with richer contextual instructions to address issues

03

Key comparison metric: number of issues identified, classified by severity levels, in both human-written and AI-refactored versions

02

Improved LLM architectures, comprehensive training data, and superior contextual understanding enable newer models to more effectively identify and resolve coding issues.

Is AI-Refactored Code Safer and Smarter? A Practical Assurance Study



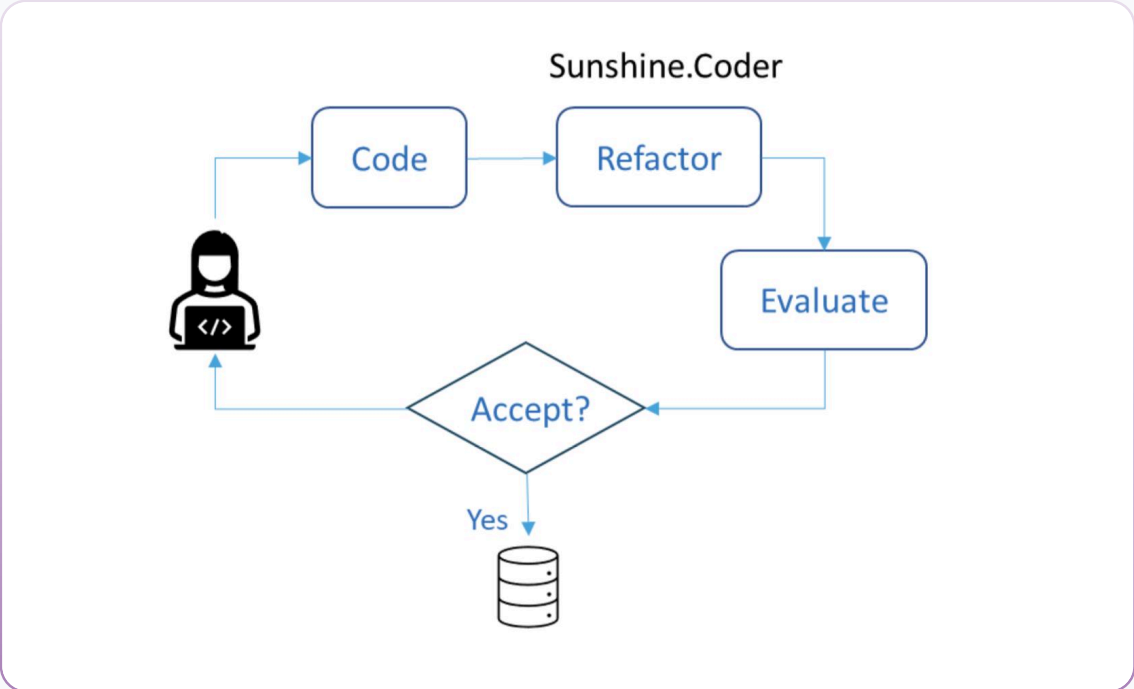
NCS is a leading technology services firm headquartered in Singapore, serving clients across the Asia-Pacific region. With deep expertise in digital transformation, cloud infrastructure, and cybersecurity, NCS continuously explores advanced technologies to enhance software engineering practices.

—

Use Case

As part of this innovation agenda, the team developed an in-house AI coding assistant called Sunshine.Coder. Designed to support enterprise-grade development workflows, Sunshine.Coder has the following key characteristics:

- 01
- It supports developers across multiple stages of the software development lifecycle, from initial implementation to post-deployment refactoring.
- 02
- It leverages large language models (LLMs) to analyse and refactor Java code, enhancing not just readability and structure, but also alignment with security, compliance, and maintainability standards.
- 03
- It comprises two components: an IDE extension for interactive, developer-in-the-loop use, and a batch processing engine for automated, large-scale code refactoring.
- 04
- It operates as a feedback-driven loop, where AI-refactored code is evaluated against acceptance criteria, and further refinements are applied until the target quality is achieved — as illustrated in Figure 1.
- 05
- The entire application runs in a secured infrastructure and is managed using a DevSecOps platform.



This pilot project aimed to evaluate Sunshine.Coder’s effectiveness in improving the quality and security of Java applications through AI-driven code refactoring, leveraging large language models (LLMs) to enhance developer productivity and assurance.

Figure 1. Iterative AI-assisted code improvement workflow for Sunshine.Coder



To independently assess the quality of AI-refactored code, NCS partnered with Parasoft, a company specialising in automated software testing, with a strong track record supporting mission-critical programs such as US DoD's Joint Federated Assurance Center (JFAC) – which promotes advanced software assurance practices across defense acquisition programs.

For this pilot, Parasoft brought deep expertise in static code analysis and applied its enterprise-grade tool, Parasoft’s JTest, to assess the Java applications refactored by Sunshine.Coder. Jtest was selected for its comprehensive capabilities in Java static analysis, code coverage and compliance checking.

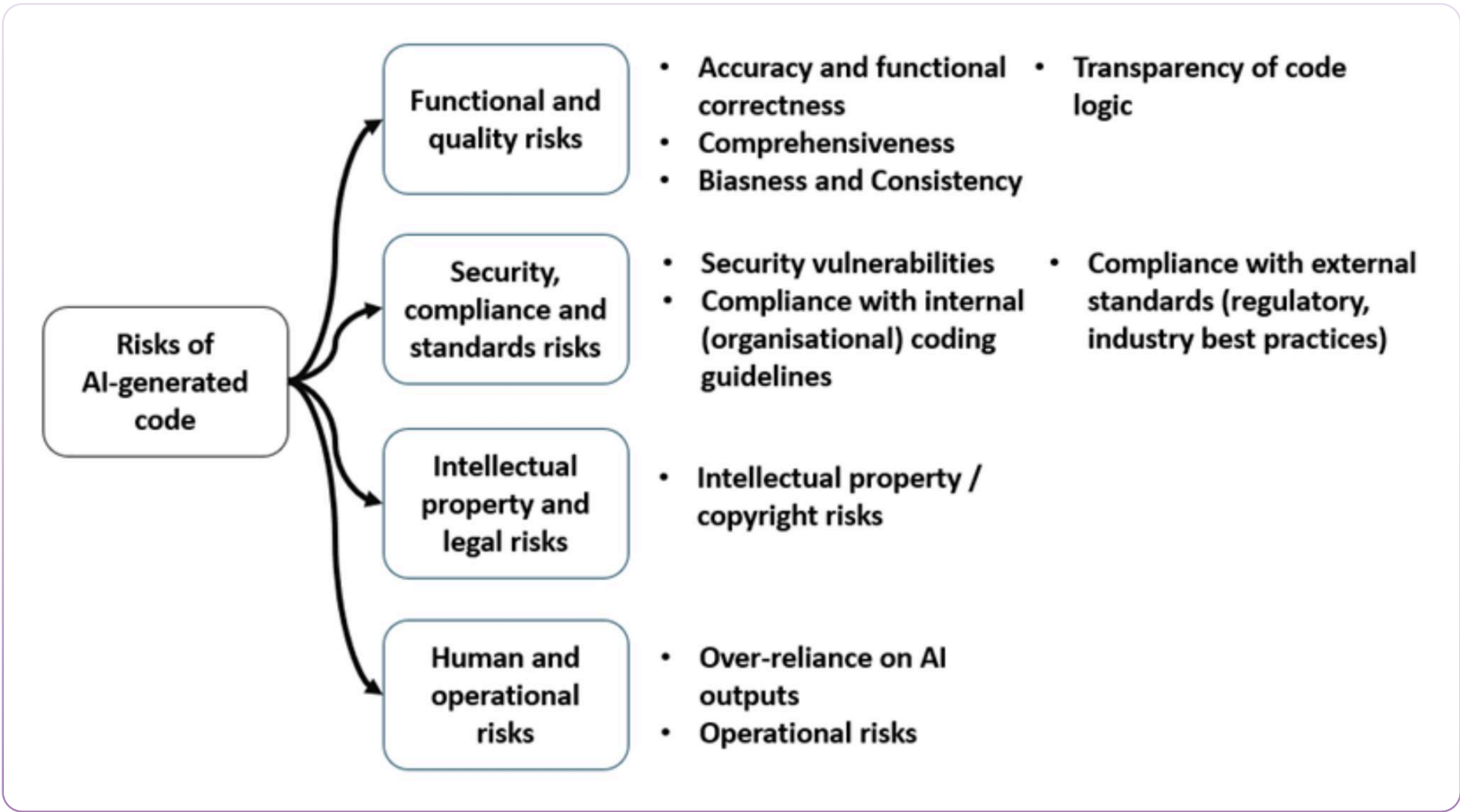


Figure 2. Risk Categories Associated with AI-Generated Code

The use of AI-driven tools to refactor human-written software code introduces a variety of risk dimensions summarised in Figure 2. For this pilot, NCS and Parasoft focused the assurance exercise on the second category: security, compliance, and standards risks. These are the most directly assessable using static analysis and the Common Weakness Enumeration (CWE) framework.

The goal was to evaluate whether AI-assisted code, when refactored by Sunshine.Coder, would introduce or mitigate common weaknesses such as security vulnerabilities, non-compliance with internal coding standards, or failure to meet external regulatory requirements.

To assess the effectiveness of AI-assisted code improvement, NCS and Parasoft designed an experiment in a secure staging environment with strict access controls around a simple but important question:

Can AI-generated code be reliably evaluated using established testing tools?

NCS’s Sunshine.Coder, an AI-powered assistant, with different LLMs and prompting strategies was used to refactor two different sets of Java code: one from a mature, production-grade system, and the other from an early-stage proof-of-concept (PoC) application.

The refactored code was then evaluated using **Parasoft Jtest**, a static analysis tool aligned with the **Common Weakness Enumeration (CWE)** — a globally recognised catalogue of software vulnerabilities maintained by MITRE. CWE provides a structured way to identify, classify, and prioritise security risks such as input validation failures or improper error handling.

With Parasoft Jtest, all code was assessed without executing the applications. Jtest findings were mapped against CWE categories and classified by severity:

- **Severity 1:** Critical issues likely to cause major security or functional failures.
- **Severity 2:** Major defects with lower immediate risk.
- **Severity 3:** Minor quality issues or standards violations.

The comparative number and severity of issues detected before and after AI refactoring served as the primary metric to assess quality improvement, degradation, or neutrality.

Two types of applications were selected to reflect different development maturity levels:

A proof-of-concept e-commerce system, likely to contain more inconsistent or risky code

A production-grade batch processing system, built with stronger adherence to coding standards

Both codebases were processed through Sunshine.Coder, using two different LLMs: one baseline model and one more advanced. We compared how each model performed when asked to improve the code, first with standard prompts and later with enhanced prompts that incorporated findings from Jtest’s security scans.

The implementation of the pilot drew on a lean but effective collaboration between NCS and Parasoft. Over the course of 8 weeks, Parasoft supported the assurance work with less than one full-time equivalent (FTE), providing advisory help to NCS to configure and execute the static analysis, run the test harness, and evaluate the AI-refactored code against industry-standard security benchmarks based on CWE.

On the NCS side, 4 developers provided part-time support throughout the pilot. Their involvement was critical in preparing the codebases, tuning the AI prompts, and interpreting results in the context of enterprise software standards. The team’s combined expertise in Java development, application familiarity, and AI-assisted programming helped ensure the testing remained relevant to real-world engineering challenges.

Despite the modest resourcing, the collaboration enabled a high-quality evaluation that demonstrated how GenAI tools could be tested securely and systematically within a DevSecOps framework.

AI Code Quality Depends on Source Code Maturity

The pilot confirmed that the quality of AI-refactored output is strongly influenced by the quality of the original human-written code. In production-grade systems, where established best practices were already in place, AI required minimal guidance to improve the code. However, proof-of-concept (PoC) code — often developed quickly and without strict standards — required richer, more context-aware prompts to achieve meaningful improvements. This demonstrates the importance of applying AI coding tools within a structured development process, especially when dealing with less mature or experimental codebases.

Stronger Models Lead to Better Refactoring Outcomes

Model capability matters. The evaluation showed that more advanced large language models, such as Qwen Coder 2.5, performed significantly better than simpler alternatives. These models were more effective in identifying and addressing critical coding issues, due to superior architecture, broader training data, and improved contextual reasoning. This highlights the importance of selecting the right model for high-assurance software engineering tasks.

Feedback from Static Analysis Enhances AI Performance

One of the most valuable findings was the impact of incorporating static analysis results directly into the AI refactoring loop. When outputs from Parasoft Jtest — based on CWE guidelines — were used as structured input to Sunshine.Coder, the AI was able to generate more precise and relevant fixes. This feedback mechanism was especially beneficial for the PoC code, improving its alignment with security and coding standards. The result is a compelling case for combining generative AI tools with traditional assurance methods to drive higher-quality outcomes.

One-Shot Prompting Has Limits

While simple prompts were sufficient for improving production-grade code, they were often inadequate for more complex or inconsistent codebases. The results reinforced that a single-pass, one-shot prompting approach has inherent limitations. More robust results were achieved through iterative workflows that integrated feedback from automated testing tools. This supports the view that AI-assisted programming should be treated as a **multi-step, feedback-driven process** rather than a one-off task.

Applying AI Intelligently Requires Engineering Discipline

Across the board, the pilot reinforced a fundamental principle: **AI tools are only as effective as the engineering workflows around them.** The ability of Sunshine.Coder to enhance code quality — including addressing CWE-classified security issues — was not just a function of the model, but of how prompts were structured, how context was provided, and how assurance tools were integrated. With the right setup, GenAI can transition from being a productivity booster to a trusted component of secure, standards-aligned software development.

As shown in Figure 3, static analysis using Parasoft Jtest plays a pivotal role at multiple points in the workflow. It first scans the original source code to identify weaknesses, then these findings are passed back to the AI system to build context-rich prompts. Sunshine.Coder refactors the code accordingly, and Jtest is triggered again to evaluate the AI-generated output. This iterative, feedback-driven process, tightly aligned with CWE standards, ensures that security is systematically improved — not assumed.

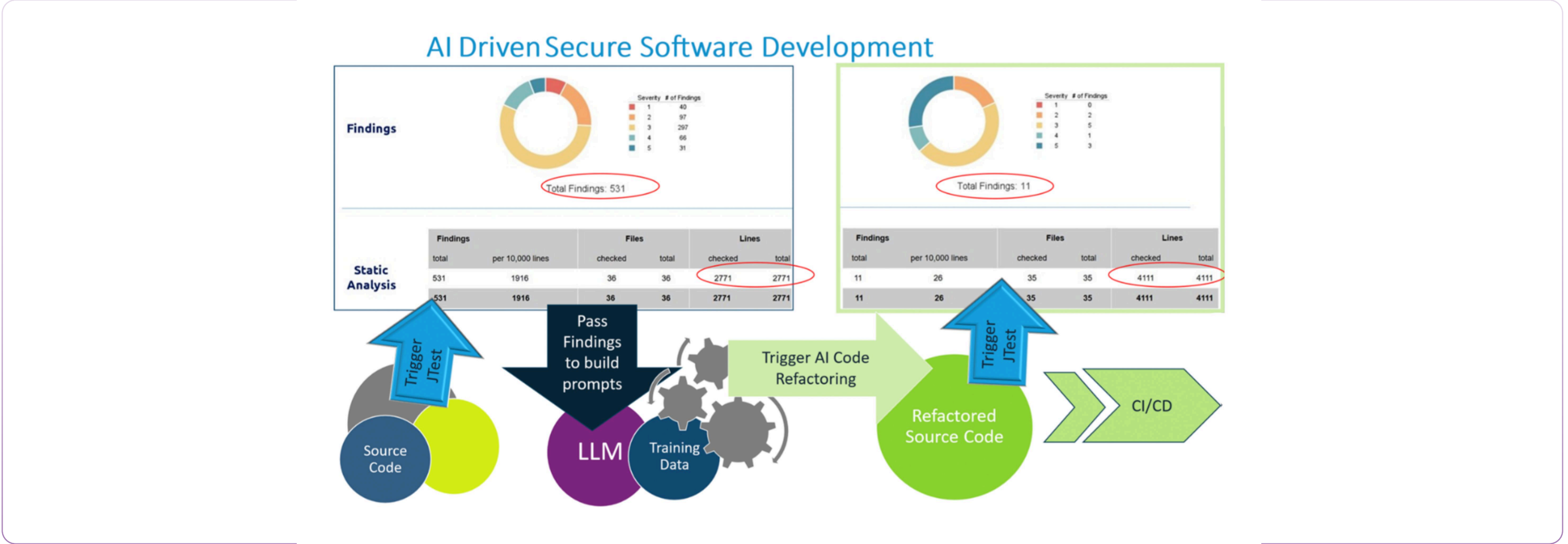


Figure 3. Workflow for AI-driven secure software development

By embedding generative AI into a secure, test-validated DevSecOps pipeline, this pilot demonstrates a scalable model for responsible and standards-aligned GenAI development. It’s not just about using AI — it’s about using it intelligently, iteratively, and with assurance at every step.