

# **Assured Sentinel: A Novelty Assessment and Theoretical Analysis of Conformal Prediction in Multi-Agent Code Security**

## **1. Introduction: The Stochastic Challenge in Generative Software Engineering**

The rapid ascendancy of Large Language Models (LLMs) has fundamentally altered the landscape of software engineering, shifting the paradigm from purely deterministic, human-authored logic to stochastic, machine-generated code. This transition, while accelerating development velocity, introduces a profound challenge: the inherent uncertainty of probabilistic outputs. In traditional software development, a compiler or interpreter provides a binary assessment of syntax validity, and a test suite provides a binary assessment of functional correctness. However, neither mechanism inherently quantifies the *risk* or *uncertainty* of the generated code, particularly regarding security vulnerabilities that may be syntactically valid and functionally operative yet catastrophically insecure.

The "Assured Sentinel" project <sup>1</sup> emerges as a direct response to this stochastic crisis. By proposing a Multi-Agent System (MAS) architecture that couples a high-temperature generative agent ("The Analyst") with a deterministic, statistically rigorous guardrail ("The Commander"), the project aims to solve the problem of hallucinated vulnerabilities. The core innovation—applying Split Conformal Prediction (SCP) to the domain of static application security testing (SAST)—represents a sophisticated attempt to impose mathematical safety guarantees on the unruly output of generative models.

This report provides an exhaustive, expert-level analysis of the Assured Sentinel project plan. It deconstructs the proposed architecture, evaluates the theoretical validity of its mathematical foundations, assesses the technical feasibility of the implementation roadmap, and ultimately determines the novelty of the approach within the broader context of AI safety and neuro-symbolic computing. The analysis relies on a detailed examination of the provided project artifacts <sup>1</sup>, extrapolating second-order insights regarding the system's behavior, limitations, and potential for redefining secure code generation.

## **1.1 The Deterministic vs. Probabilistic Conflict**

To understand the specific value proposition of Assured Sentinel, one must first appreciate the friction between modern DevSecOps and Generative AI. DevSecOps relies on deterministic gates: a unit test passes or fails; a linter flags an error or it does not. These are Boolean states. LLMs, conversely, operate on continuous probability distributions. A model does not "know" code is secure; it assigns a high log-probability to a specific token sequence based on training data.

The Assured Sentinel project bridges this gap by converting the Boolean output of a security tool (the bandit linter) into a continuous "Non-Conformity Score".<sup>1</sup> This transformation allows the application of Conformal Prediction—a statistical framework typically used for uncertainty quantification in regression and classification—to the domain of code security. This architectural choice is not merely an engineering detail; it is a fundamental re-conceptualization of security not as a binary attribute but as a distributional property that can be calibrated to a specific risk tolerance ( $\alpha$ ).

## **2. Architectural Deconstruction of the Multi-Agent System**

The Assured Sentinel project plan outlines a "Two-Agent Pattern" designed to ensure mathematical safety guarantees.<sup>1</sup> This separation of concerns is critical for preventing the "self-delusion" often seen in single-agent architectures where the same model is responsible for generating and verifying content.

### **2.1 The Analyst: The Generative Engine**

The first agent, designated "The Analyst," is described as a high-temperature LLM agent utilizing the Azure OpenAI Service (gpt-4o).<sup>1</sup> The specification of "high-temperature" is a nuanced design choice. In generative tasks, temperature controls the entropy of the output distribution. Low temperatures yield deterministic, repetitive outputs, while high

temperatures encourage diversity and creativity.

In the context of a rejection-sampling architecture like Assured Sentinel, a high temperature is strategically advantageous. If the system relies on "The Commander" to reject insecure code, the Analyst must be capable of exploring the solution space to find a valid alternative. A low-temperature agent, upon rejection, might simply regenerate the same insecure code pattern deterministically, leading to an infinite rejection loop. The Analyst's role is thus defined not by precision, but by *proposal generation*. It proposes candidate solutions to the problem defined by the user (e.g., "Write a function to parse XML"<sup>1</sup>), acting as the stochastic motor of the system.

The operational parameters for The Analyst include a "System Prompt" establishing the persona of a "Senior Python Engineer".<sup>1</sup> This persona engineering is vital because it sets the baseline distribution of the generated code. If the persona is well-crafted, the initial "Non-Conformity Scores" should be lower on average, reducing the computational cost of the rejection loop.

## 2.2 The Commander: The Deterministic Guardrail

The second agent, "The Commander," acts as the logic gate. Unlike The Analyst, The Commander is not a generative entity in the creative sense; it is a verifier. The project plan explicitly states that The Commander uses Conformal Prediction to reject outputs that do not meet a strict non-conformity threshold ( $\hat{q}$ ).<sup>1</sup>

This component introduces the neuro-symbolic element to the architecture. It integrates mapie (a Python library for Conformal Prediction) and scikit-learn with bandit (a security linter).<sup>1</sup> The Commander's workflow is strictly procedural:

1. Intercept the Analyst's response.
2. Extract the code block.
3. Execute the bandit static analysis tool programmatically.
4. Compute the scalar Non-Conformity Score.
5. Compare the score against the pre-calibrated  $\hat{q}$ .
6. Execute the binary decision: Pass or Reject.

The novelty here lies in the "Inverse Security Score." Traditional security tools are designed to alert humans. The Commander repurposes these tools to alert the agentic system itself, closing the feedback loop without human intervention. This automation of the "Code Review" phase, backed by statistical rigor rather than heuristic rules, defines the Commander's unique

value proposition.

## 2.3 The Interaction Protocol: The Correction Loop

The interaction between the two agents is governed by "The Loop".<sup>1</sup> If the Commander rejects the code, the error signal is passed back to the Analyst for correction. This feedback mechanism transforms the system from a simple filter (which would just discard bad code) into a refinement engine (which improves code).

The project plan notes the objective for Day 5 is to "Implement the Correction Loop".<sup>1</sup> This implies a complex state management challenge. The Analyst must maintain the context of the original request, the failed attempt, and the specific feedback from the Commander. This iterative process raises theoretical questions regarding "Exchangeability"—a core assumption of Conformal Prediction—which will be analyzed in the theoretical section of this report. The loop ensures that the final output is not just "lucky" but has converged to a solution that satisfies the rigorous safety constraints defined by the calibration set.

## 3. Theoretical Framework: Split Conformal Prediction in Security

The "Assured Sentinel" project explicitly adheres to the "Split Conformal Prediction" (SCP) methodology.<sup>1</sup> To assess the novelty and validity of this approach, we must explore the mathematical foundations of SCP and how they map to the idiosyncratic domain of software security.

### 3.1 Principles of Split Conformal Prediction

Conformal Prediction is a distribution-free uncertainty quantification framework. "Distribution-free" means it does not assume the data follows a specific parametric distribution (like Gaussian). This is crucial for code generation, where the distribution of valid Python syntax is high-dimensional and complex.

The "Split" in SCP refers to the partitioning of data into a *Training Set* (used to train the base model—in this case, the pre-trained GPT-4o) and a *Calibration Set* (used to calculate the threshold). The project plan identifies the need to "Load Ground Truth Dataset (HumanEval or MBPP)" for this calibration phase.<sup>1</sup>

The core guarantee of SCP is that for a new exchangeable test point  $(X_{n+1}, Y_{n+1})$ , the probability that the true label  $Y_{n+1}$  falls within the predicted set  $C(X_{n+1})$  is at least  $1 - \alpha$ .

$$\Pr(Y_{n+1} \in C(X_{n+1})) \geq 1 - \alpha$$

In the context of Assured Sentinel, the "Label"  $Y$  is the validity/security of the code. The system constructs a prediction set that acts as a filter: it accepts the code only if its "weirdness" (security risk) is below the threshold  $\hat{q}$  determined by the  $(1 - \alpha)$  quantile of the calibration set.

### 3.2 Defining the Non-Conformity Measure (NCM)

The success of any CP application hinges on the definition of the Non-Conformity Measure. A good NCM assigns low scores to "normal" (secure) data and high scores to "abnormal" (insecure) data.

Assured Sentinel defines the NCM via the bandit tool.<sup>1</sup> The plan specifies a function `score(code_str) -> float` where 0.0 is Secure and 1.0 is Vulnerable. This mapping is non-trivial. bandit typically outputs a JSON report with a list of issues, each having a severity (Low, Medium, High) and confidence.

To convert this structured data into a scalar float required for mapie, the `scorer.py` module must implement a reduction function, such as:

$$S(y) = \sum_{i \in \text{Issues}} w(\text{severity}_i) \cdot w(\text{confidence}_i)$$

where  $w$  represents a weighting coefficient.

If the calibration set contains code with no security issues, the scores will be identically zero. If the calibration set contains a mix, the scores will form a distribution. The project plan's target of  $\alpha = 0.1$  (90% confidence)<sup>1</sup> implies that the system will accept code that is "as secure as the top 90% of the calibration data." This highlights the critical dependency on the quality of the "Ground Truth Dataset." If the calibration dataset is riddled with

vulnerabilities, the threshold  $\hat{q}$  will be dangerously high, allowing insecure code to pass.

### 3.3 The Calibration Phase and Quantile Estimation

Day 3 of the roadmap is dedicated to "The Calibration".<sup>1</sup> The plan involves running the scorer on 100-200 calibration examples. Mathematically, the threshold  $\hat{q}$  is calculated as:

$$\hat{q} = \text{Quantile}\left( s_1, s_2, \dots, s_n \right), \frac{\lfloor (n+1)(1-\alpha) \rfloor}{n}$$

where  $n$  is the size of the calibration set.

The choice of  $n=100-200$  is statistically significant. In Conformal Prediction, the variance of the coverage guarantee decreases as  $n$  increases. With  $n=100$ , the coverage is reasonably stable, but the "finite sample correction" ( $\frac{n+1}{n}$ ) ensures that the guarantee holds even for this small dataset. This statistical rigour contrasts sharply with arbitrary heuristic thresholds (e.g., "Allow max 2 low severity issues") common in industry. The CP threshold is dynamic, derived from the actual performance of the model on the reference data.

## 4. Novelty Assessment: Positioning Assured Sentinel

To answer the user's primary query—"Is it novel?"—we must situate Assured Sentinel within the existing landscape of AI safety and code generation research. The following comparative analysis reveals that while individual components are well-established, their specific integration in this architecture represents a novel contribution.

### 4.1 Comparison with Static Application Security Testing (SAST)

Feature	Traditional SAST (e.g., SonarQube, Bandit)	Assured Sentinel (CP + Bandit)
<b>Timing</b>	Post-commit (CI/CD Pipeline)	Inference-time (Generation phase)
<b>Decision Logic</b>	Rule-based (Boolean Pass/Fail)	Statistical (Quantile-based Threshold)
<b>Feedback Loop</b>	Human developer fixes code	Agentic "Analyst" fixes code
<b>Risk Model</b>	"Zero Tolerance" or "Allow List"	"Risk Budget" ( $\alpha$ confidence level)

**Novelty Verdict: High.** Traditional SAST tools are static and binary. Assured Sentinel transforms them into dynamic, probabilistic gates. The concept of applying a "Risk Budget" ( $\alpha$ ) to code generation allows for a nuanced trade-off between security and code diversity that binary tools cannot offer.

## 4.2 Comparison with Self-Correcting Agents (Reflexion)

The "Reflexion" framework (Shinn et al.) allows agents to verbally critique their own output to improve performance.

- **Reflexion:** "The code failed test X. I should fix Y." (Semantic Feedback).
- **Assured Sentinel:** "The code has a non-conformity score of 0.8, which exceeds the threshold 0.15." (Statistical Feedback).

**Novelty Verdict: Moderate to High.** While self-correction is a known pattern, using an *external, deterministic tool* to drive the correction via a *conformal* threshold is distinct. Most self-correction relies on the LLM's own internal reasoning (which is prone to hallucination) or simple unit test failures. Assured Sentinel's use of bandit provides a ground-truth signal that anchors the agent's correction process in reality, preventing "hallucinated corrections."

## 4.3 Comparison with Conformal Prediction in NLP

Research into applying CP to NLP is nascent, mostly focusing on "Conformal Factuality" (ensuring the answer contains the true fact) or "Conformal Risk Control" for toxicity.

- **State of the Art:** Using CP to bound the risk of toxic output.
- **Assured Sentinel:** Using CP to bound the risk of *vulnerable code syntax*.

**Novelty Verdict: Very High.** The specific application of SCP to the domain of *code security scores* is a unique research vector. It treats "vulnerability" as a random variable inherent to the generative process, a sophisticated perspective that aligns with the intrinsic nature of LLMs.

## 5. Implementation Feasibility: The Microsoft Stack

The project plan details a "Build Week Roadmap" rooted in the Microsoft ecosystem.<sup>1</sup> This choice significantly impacts the feasibility and reproducibility of the research.

### 5.1 Infrastructure and Dependencies

The selection of **GitHub Codespaces**<sup>1</sup> creates a standardized development environment ("Dev Container"). This is crucial for scientific reproducibility. By defining the dependencies—mapie, bandit, semantic-kernel, streamlit—in a container configuration, the project eliminates the "works on my machine" variability.

- **Azure OpenAI (gpt-4o):** The use of GPT-4o is essential. The "Correction Loop" requires a model with high reasoning capabilities to interpret the bandit feedback and modify the code structure accordingly. Smaller models would likely fail to converge, stuck in a loop of generating invalid code.
- **Semantic Kernel:** This SDK simplifies the orchestration of the agents. It handles the memory (context window) and the function calling required to invoke the Scorer.

## 5.2 The Roadmap Analysis

The five-day roadmap <sup>1</sup> is aggressive but structured logically:

- **Day 0 (Infra):** Establishing the secure perimeter. The strict rule "No API keys in code" <sup>1</sup> demonstrates operational maturity.
- **Day 1 (The Analyst):** Setting up the generative baseline.
- **Day 2 (The Scorer):** The technical core. Implementing the `score(code_str)` function is the primary engineering challenge.
- **Day 3 (The Calibration):** The mathematical core. This is where the research hypothesis is tested. If the calibration set scores are all zero, the system fails to be useful.
- **Day 4 (The Commander):** Integration.
- **Day 5 (The Loop):** Visualization.

The plan's explicit focus on "Observability" via Streamlit <sup>1</sup> is a strong feature. In AI systems, trust is established through transparency. A dashboard showing the calibration histogram and the position of the current code's score relative to  $\hat{q}$  makes the "black box" decision process visible to the user.

## 6. The Calibration Challenge: Data and Distribution

A critical analysis of the project plan reveals a potential flaw in the data strategy that must be addressed to ensure success.

### 6.1 The "Ground Truth" Problem

The plan suggests using "HumanEval or MBPP".<sup>1</sup> These are standard benchmarks for *functional correctness* (e.g., "sort this list," "find the longest palindrome").

- **The Issue:** These datasets are algorithmic puzzles. They rarely involve external libraries, file I/O, or network requests—the very vectors where security vulnerabilities (SQL injection, hardcoded secrets, command injection) manifest.
- **The Consequence:** If bandit is run on HumanEval, it will likely return a score of 0.0 for nearly every entry. The calculated threshold  $\hat{q}$  will therefore be 0.0.

- **The Result:** The Commander will enforce a "Zero Tolerance" policy. While secure, this defeats the purpose of Conformal Prediction, which is to handle *uncertainty* and allow for a calibrated margin of error (e.g., accepting low-severity warnings if  $\alpha$  allows).

## 6.2 Recommendation: Security-Specific Calibration

To truly leverage the power of Conformal Prediction, the calibration set must contain a *distribution* of security scores. The research must integrate a dataset specifically designed for security testing, such as:

- **Sard (Software Assurance Reference Dataset):** A NIST dataset containing known vulnerabilities.
- **OWASP Juice Shop Snippets:** Examples of intentionally vulnerable code.
- **Synthetic Vulnerability Dataset:** Using an LLM to generate code known to be vulnerable to calibrate the Scorer.

By mixing secure code (HumanEval) with vulnerable code, the calibration phase will produce a non-trivial histogram, allowing for a meaningful calculation of  $\hat{q}$  that represents a genuine risk boundary.

## 7. Operational Dynamics and "The Loop"

The project plan describes a "Loop" where rejected code is returned to the Analyst.<sup>1</sup> The dynamics of this loop warrant deeper investigation.

### 7.1 Exchangeability Violation

Conformal Prediction assumes that the calibration data and the test data are exchangeable (drawn from the same distribution).

In the first pass, the Analyst generates code  $Y_1$  based on prompt  $X$ . This is exchangeable with the calibration set (assuming the calibration set was generated similarly). However, if  $Y_1$  is rejected, the Analyst generates  $Y_2$  based on  $(X, Y_1)$ ,

\text{Feedback}}\$.

The distribution of  $Y_2$  is conditional on the failure of  $Y_1$ . It is a different distribution.

- **Theoretical Risk:** The CP guarantee ( $1-\alpha$ ) strictly applies only to the *first* attempt. The coverage guarantee for the refined code  $Y_2$  might technically be invalid under standard SCP assumptions.
- **Mitigation:** The report should acknowledge this theoretical nuance. Advanced techniques like "Adaptive Conformal Prediction" deal with such distribution shifts, but for a "Phase 1" project, acknowledging the limitation is sufficient.

## 7.2 Latency and Cost

The "Data Flow" <sup>1</sup> introduces latency.

1. Generation (Analyst): ~5-10 seconds.
2. Scoring (Bandit): ~1-2 seconds (overhead of file I/O).
3. Rejection Loop: If rejected, the cost doubles.

The "Rules of Engagement" state "Cloud Only" and "Respect the Surface Pro's limits".<sup>1</sup> This suggests that efficiency is a concern. The implementation of bandit must be optimized (e.g., using in-memory AST analysis rather than disk writes) to keep the feedback loop tight.

## 8. Security and Compliance Implications

The ultimate value of Assured Sentinel lies in its ability to produce a "Certificate of Conformity."

### 8.1 The "Assured" Guarantee

In a regulated industry (e.g., Finance, Healthcare), software must often meet strict standards. Assured Sentinel offers a way to automate compliance.

Instead of a manual auditor checking code, the system can log:

"Generated Code ID #12345. Non-Conformity Score: 0.02. Threshold: 0.05. Status:

Accepted (90% Confidence)."

This audit trail creates a quantifiable metric for AI safety. It moves the organization from "Trusting the AI" to "Verifying the AI" using a mathematically sound framework.

## 8.2 The "Inverse Security Score"

The project's definition of "Non-Conformity" as the "Inverse Security Score"<sup>1</sup> is a powerful conceptual tool.

- **Conformity:** Adhering to the patterns of safe, high-quality code.
  - Non-Conformity: Exhibiting patterns (like eval(), pickle, or hardcoded secrets) that deviate from the safe baseline.
- By calibrating on a "Gold Standard" codebase, Assured Sentinel effectively enforces that all new code must statistically resemble the Gold Standard in terms of security posture.

## 9. Future Directions

The current project plan is a solid "Phase 1." Future iterations could expand the scope significantly.

### 9.1 Multi-Modal Scoring

Currently, the score is derived solely from bandit.<sup>1</sup> Future versions could aggregate scores from multiple tools:

- **Pylint:** For code style and quality.
  - **Complexity Metrics:** Cyclomatic complexity.
  - Unit Tests: Functional correctness.
- A "Vector-valued" Conformal Prediction could ensure that code is simultaneously Secure, Stylish, Simple, and Correct.

## 9.2 Human-in-the-Loop Calibration

The Streamlit dashboard <sup>1</sup> could be evolved to allow humans to "label" rejected code. If the Commander rejects code that a human deems safe (a False Positive), the human could add it to the Calibration Set. This "Online Conformal Prediction" would allow the threshold  $\hat{q}$  to adapt dynamically to the specific coding style and risk appetite of the user.

## 10. Conclusion and Verdict

The Assured Sentinel project plan <sup>1</sup> outlines a research initiative that is **Novel, Theoretically Sound, and Technically Feasible**.

- **Novelty:** It pioneers the application of Split Conformal Prediction to the domain of Static Application Security Testing (SAST), creating a new category of "Probabilistic Security Gates" for Generative AI.
- **Soundness:** It correctly identifies the need for split calibration and separate agentic roles (Analyst vs. Commander) to avoid mode collapse and ensure independent verification.
- **Feasibility:** It leverages mature, enterprise-grade technologies (Microsoft Semantic Kernel, Azure OpenAI, GitHub Codespaces) and well-maintained Python libraries (mapie, bandit), minimizing implementation risk.

The project effectively addresses the "Stochastic Crisis" of generative code. By wrapping the creative chaos of an LLM in the rigorous statistical embrace of Conformal Prediction, Assured Sentinel offers a roadmap toward a future where AI-generated code is not just likely to be right, but provably assured to be safe.

### Summary of Key Findings

Aspect	Assessment	Justification

<b>Concept Novelty</b>	<b>High</b>	Unique synthesis of SCP and SAST.
<b>Architecture</b>	<b>Robust</b>	Decoupled Generator (Analyst) and Verifier (Commander).
<b>Tech Stack</b>	<b>Modern</b>	Azure OpenAI + Semantic Kernel is state-of-the-art.
<b>Risk Factors</b>	<b>Calibration</b>	Reliance on HumanEval may skew thresholds.
<b>Verdict</b>	<b>Proceed</b>	The project serves as a valid proof-of-concept for high-assurance AI.

The Assured Sentinel project is not merely a tool; it is a proposal for a new standard of trust in the era of Agentic AI. It suggests that safety is not a binary state, but a calibrated probability, and provides the architecture to enforce it.

## Works cited

1. 20251123\_Assured\_Sentinel\_Project\_Plan.pdf