SentinelAI: Combining Static Analysis and Retrieval-Augmented Reasoning for Explainable Vulnerability Detection

Tanaya Paralkar[1], Nitaanth Panickar[1], Viraj Darekar[1], and Jaya Jeswani[1]

[1] Department of Data Science, Mukesh Patel School of Technology Management and Engineering, NMIMS, Mumbai, Maharashtra, India
{tanayaparalkar, nitaanth.panickar, vrjdarekar}@gmail.com,
jaya.jeswani@nmims.edu

Abstract. Static application security testing tools give crisp and specific results, but using multiple of them leaves organizations with disjointed reporting, inconsistent severity language and not knowing how to fix it. In this research, the authors introduce SentinelAI, a command-line DevSecOps system that integrates multi-scanner execution, typed normalization, deterministic aggregation, provenance-preserving reporting, and optionally a fully local large language model (LLM). SentinelAI comes in two levels: core (Semgrep, Bandit, GitLeaks) and extended (Trivy, OSV-Scanner, recursive manifest discovery). The raw scanner results are normalized and grouped by normalized CWE, file identity, and a two-line location tolerance. The core tier found 23 raw results and 16 correlated groups on the tested fixture, the extended tier 60 raw results and 53 correlated groups, and the deterministic grounding verifier rejected 7 of 16 AI enrichments. No assertions are made about precision, recall, calibration, or utility in the research.

Keywords: DevSecOps, Static application security testing (SAST), Vulnerability scanning, Finding correlation, SARIF, Retrieval-augmented generation (RAG), Local large language models, Reproducible tooling, Fail-closed design.

1 Introduction

The changes have not been easy to make in practice, which is not what the concept suggests. Multiple analyzers are required for satisfactory coverage, but break up the results. The output format, severity scale, rule set and weakness taxonomy are different among each analyzer, and overlaps should be reconciled by the developers.

The secondary issue is volume. In five large projects, only 27.4% to 49.5% of static-analysis alerts were actionable, and developers were slow to fix them even when the fix was simple [1]. Alike repercussions are reported for default analyzer configurations [2] and for suppressed warnings [3]. Detection capacity is not a constraint but the route from detection to resolution is.

Language models are a natural fit because they are missing an element – an explanation – not another detector. Models fail to reason reliably about vulnerabilities once superficial cues are removed [4] and benchmarks' performance declines substantially when benchmark artefacts are removed [5, 6]. Retrieval-augmented reasoning has proven to enhance grounding [7] and has been used to

reason about vulnerabilities [8] but is not used to make generated text authoritative. Two design temptations are avoided: suppression buys silence at the cost of recall, and ungrounded generation produces narrative that is answerable to no particular fact.

This paper is an engineering validation and architecture study, not a study of detection effectiveness. It describes the system, shows that its behavior is repeatable, and verifies that its invariants hold at run time. No claims are made here regarding the detection effectiveness, calibration or alert fatigue as there was no labelled corpus to make the claims; these limits are explicitly stated in Sections 5.1 and 7. There are 5 contributions: (C1) add a correlation scheme that records the agreement between the two scanners, while preserving all of the original findings; (C2) add a persistence contract, that keeps the findings separate from the interpretation; (C3) add a fail-closed contract, that treats failures as infrastructure issues, not failures or a non-event; (C4) add an optional local enrichment path, that checks whether an interpretation names the weakness the scanner reported; and (C5) add a packaging discipline for reproducibility, that pins the version of each scanner, and that packages the three as release binaries, which are additionally checked against the publisher's SHA-256 at image build time.

2 Background and Related Work

2.1 Positioning Against Contemporary Platforms

Existing aggregation platforms converge on deduplication as the way to control overlap: DefectDojo marks duplicates inactive against a retained original, GitHub Code Scanning matches results by `partialFingerprints`, and the Semgrep AppSec Platform deduplicates across scans. SentinelAI differs by correlating additively: every raw finding from its five scanners in two tiers is preserved and groups are added, not substituted. The SARIF 2.1.0 output keeps model text in the property bag and provides an additional distinction for a policy outcome from a tool failure, which Semgrep's CLI already does for its own scans, and which SentinelAI applies during a multi-scanner run. Enrichment is optional and entirely local, checked by a deterministic lexical verifier, Semgrep Assistant and Copilot Autofix are reliant on hosted LLM services, and Copilot Autofix uses a second LLM as its filter. SentinelAI lacks in depth of scanners: five fixed scanners vs DefectDojo's parser library plus SonarQube's external-analyzer list; additive correlation means that every raw finding is kept, which leads to fewer groups to review, but not fewer findings stored; and its validated remediation neither re-runs analysis in CI nor opens pull requests.

SentinelAI's contribution is neither a new detector, nor a new use of an LLM. There are numerous scanners that are already being ingested by aggregation platforms, and commercial tools already attach model generated fixes. What is different is a set of contracts that exist between evidence and interpretation. Deduplicated findings over platforms, surveyed platforms do not discard the findings, but combine them into groups. There is no message field in which models can be entered into the scanner; they only reside in the SARIF property bag. Failure of a scanner won't be an indication of a policy failure for the entire

multi-scanner run. In a sense, enrichment is completely local; that is, this is not checked by another model, as in IRIS. Each of these is little more than a humdrum statement, and the paper's claim is that combination — which it is not testing for detection accuracy, but for invariance.

3 System Design and Architecture

3.1 Design Principles

The design is based on 5 principles. Principle 1: Separation of provenance: deterministic evidence and model interpretation are objects that share an identity, but are not merged, and are independently validated. Principle 2: Additive correlation: structure of findings is not lost, but enhanced by grouping. Principle 3: Fail closed operation: In case of a scanner failure, it does not affect the results of the scanner. Principle 4: Determinism at the boundaries: If given identical inputs, then identical group membership, identifiers, and serialization. Principle 5: Optionality: reasoning is another layer – it is activated only when it is desired.

3.2 Pipeline Overview

Fig. 1 shows the pipeline. When enrichment is requested, each affected issue passes through retrieval, reasoning, and a verification step, and the resulting record carries the separate observations and renders as any of five views. Without enrichment the record moves from correlation to presentation unchanged except for the absent advisory fields.

3.3 Typed Contracts and Additive Correlation

Validated schema objects transport information for layers. Records emitted by a scanner are added to a scanner finding, a set of normalized classification and a path or a line, etc., are added to a correlated finding, and an interpretation and a confidence value of a model are added to an enriched finding, and a verification status is bound to a finding it interprets. There are two requirements that follow, which are enforced by the code, not just documented. An enriched finding cannot exist without a corresponding scanner finding, so nothing is interpreted that no scanner reported. A scanner that fails to run raises an operational error instead of contributing a finding, which is what prevents a broken tool chain from presenting itself as a clean repository.

Each scanner wrapper reads the weakness identifier from the scanner's own output: Semgrep from `metadata.cwe`, Bandit from `issue cwe`, Trivy from `CweIDs`, and OSV-Scanner from `database specific.cwe ids`. GitLeaks reveals none. A single pattern, `CWE-(\d+)` [10], normalizes these to the form `CWE-<n>`, and where a scanner lists several identifiers, the first is used. By construction, a result whose file field is empty, or whose line field is, or whose CWE field is, is a singleton. Two correlatable findings are linked when they share a file and a normalized CWE and the gap between their line ranges is at most two lines, where overlapping or touching ranges have a gap of zero. The linkage property of connectedness is transitive: if the first two points are connected and the last two points are connected, then the points in between are connected. This is done by the use of union-find on the pairwise relation (Algorithm 1),

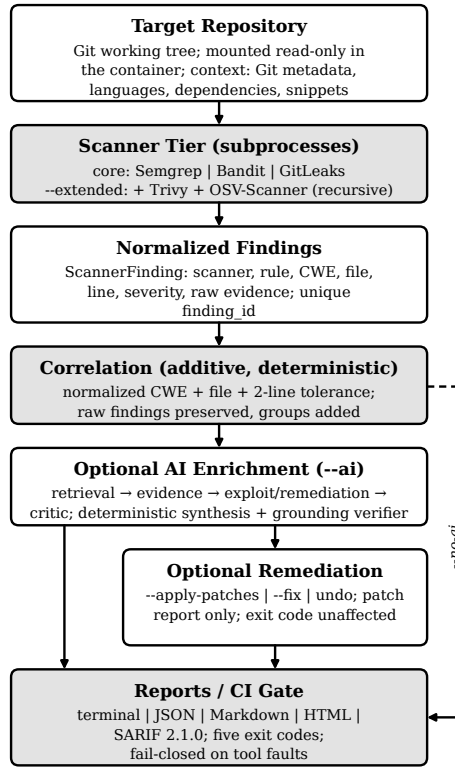


Fig. 1: The Layered pipeline of SentinelAI . Enrichment and remediation are optional, structurally subordinate to scanner evidence. The dashed edge is the `--no-ai` path, which skips enrichment.

which allows group membership to be independent of the order of comparison. Group identifiers are assigned sequentially (`CORR-001`, `CORR-002`, ...) over the sorted groups, so similar inputs yield identical identifiers. Raw finding identifiers originate from the scanner's own fingerprint where one exists (GitLeaks, Semgrep), from a positional counter (Bandit), or from a composite of manifest, package, version, and advisory (Trivy, OSV-Scanner).

3.4 Optional Grounded Enrichment

Enrichment is based on correlated issues and not on raw evidence, so the path of enrichment is a weakness, rather than an individual alert; Fig. 2 illustrates the path of one issue. Entries are then retrieved from a small curated knowledge base for each issue, and then passed through three passes: evidence analyst determines what the raw evidence implies, exploit and remediation analyst infers what conditions would make the evidence exploitable and what remediations are available, and a critic argues against the evidence, the conditions for the exploit, and the available remediation. The three responses are combined by an ordinary code called a deterministic synthesizer and passed to grounding verification.

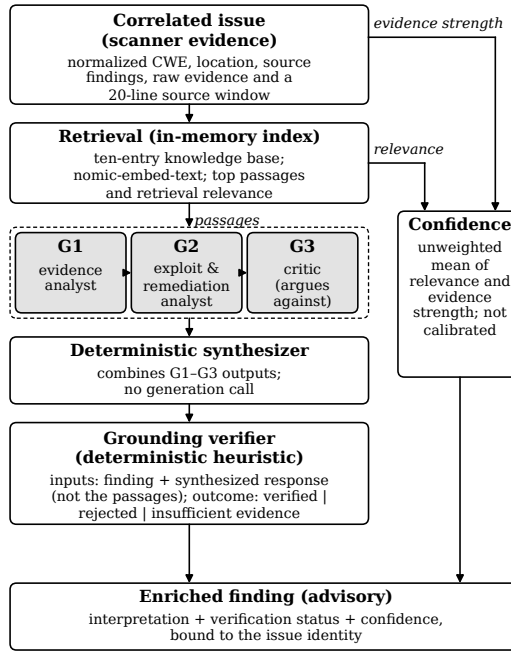


Fig. 2: Enrichment of a single correlated issue that is grounded. Only local model calls are shaded stages, the rest of the code is ordinary deterministic code, and the verifier never sees the passages retrieved.

Algorithm 1: Additive correlation of scanner findings

Input: *raw findings F, line tolerance t = 2*
Output: *groups G; every member of F still reachable*

sort F by (file, line, normalized CWE, finding id)
for each f in F: **if** f has no file, no line, or no CWE, **mark** f singleton
for each pair (a, b) of non-singleton findings:
    if same file and same normalized CWE and gap(a, b) <= t:
    union(a, b)
G := connected components of the union-find structure
sort G by (file, line, CWE, canonical finding id)
for i = 1 .. $|G|$: id(`g_i`) := `"CORR-"` + zero-padded i
return G together with the unmodified F

A validator is a deterministic lexical heuristic. The synthesized response is combined with the report of the scanner finding and never sees the retrieved passages which are general knowledge, and therefore cannot be used as a ground truth for a claim about a specific finding. If there is no raw-evidence excerpt in the finding or the category is empty after normalization, then it returns insufficient evidence. If not, then the scanner's weakness category is normalized (lower case, hyphens and underscores replaced by spaces, generic words like injection, exposure, and traversal removed). If the normalized category is in the

concatenated explanation, remediation, exploit-path, and impact text, or if any non-generic word of the category is, then the response is verified, otherwise it is rejected. The check thus checks that the response contains the name of the weakness reported by the scanner.

It does not compare the response with the raw evidence, and it does not assess correctness. No finding on the fixture reached the insufficient-evidence branch. The confidence score derives from two inputs alone, retrieval relevance and evidence strength, combined as an unweighted mean; it is therefore neither a model self-report nor a calibrated quantity.

3.5 Fail-Closed Operational Contract

A list of the five ending conditions is given in Table 1. The major difference between code 1 and code 3 is that code 1 uses the same code as the original record, whereas code 3 is a different code. If a non-zero scanner exit is detected, it will report an operational error and terminate

Table 1: Exits and exit codes for a pipeline.

Code	Terminating condition	Significance for the calling pipeline
0	Scan completed; nothing crossed the configured threshold	Proceed
1	Scan completed; at least one finding crossed `--fail-on`	Policy decision: a security outcome
2	Invalid user input	Operator error; the invocation is wrong
3	Scanner or provider failure	Infrastructure fault; never a clean result
4	Internal failure	Defect in SentinelAI itself

4 Implementation

4.1 Packaging, Dependencies, and Scanner Wrappers

Table 2 gives each invocation and the severity mapping. Scanners run with their built-in default rules; no custom rule file or configuration is supplied. GitLeaks reports no severity, so every secret finding is assigned high. No scanner subprocess has a timeout.

Table 2: Scanner invocation and mapping of native severity to SentinelAI severity (unrecognized or missing values map to low).

Scanner (tier)	Invocation	Version	Severity mapping
Semgrep (core)	`semgrep --json --quiet <repo>` (no `--config`)	1.172.0	ERROR→high, WARNING→medium, INFO→low
Bandit (core)	`bandit -r -f json <repo>`	1.9.4	HIGH, MEDIUM, LOW kept as is
GitLeaks (core)	`gitleaks dir <repo> -f json -r - --redact`	8.30.1	every finding→high
Trivy (ext.)	`trivy fs --scanners vuln --format json <repo>`	0.74.0	CRITICAL, HIGH, MEDIUM, LOW kept as is
OSV-Scanner (ext.)	`osv-scanner scan source --format json --recursive <repo>`	2.5.1	max CVSS $\geq$9 critical, $\geq$7 high, $\geq$4 medium, else low

4.2 Knowledge Base, Retrieval, and Reasoning

The knowledge base holds ten entries covering ten weakness categories: SQL injection, command injection, cross-site scripting, path traversal, insecure deserialization, broken access control, weak cryptography, secret exposure, misconfiguration, and vulnerable dependency (Table 3).Every Knowledge-base entry includes a category, vulnerable pattern, an exploit condition, remediation advice, a CWE identifier and reference URLs. Only the category and pattern, condition and guidance are embedded and passed to the model. The CWE identifier and references are not embedded, are not in any prompt, and are metadata, not read by the verifier. References are attached to the output deterministically after generation, so the model cannot introduce a citation.

Table 3: Knowledge base: ten entries and their CWE identifiers.

Category	CWE	Category	CWE
SQL injection	89	Broken access control	862
Command injection	78	Weak cryptography	327
Cross-site scripting	79	Secret exposure	798
Path traversal	22	Misconfiguration	489
Insecure deserialization	502	Vulnerable dependency	1395

Enrichment is done with open-weight models to which the locally available Ollama: `llama3.1:8b` (instruction-tuned, 8 billion parameters) is used for generation and Ollama:`nomic-embed-text` for embeddings. Each passage in the knowledge-base will be scored using cosine similarity between their embeddings in an in-process index. The five highest-scoring passages are passed to each generation stage, and the retrieval-relevance signal is the score of the top-ranked passage. Temperature: 0.0 - the same as Ollama's default. No seed, top-p, or context-size option is set, so Ollama's defaults are used. Ollama's `format` parameter limits output to a per-stage JSON schema, and through strict typed models, it rejects extra fields, as validation. The timeout for each of the requests is 120 s. The requests are retried only once after a two-second delay, for connection failures and transient server errors (HTTP 500, 502, 503, 504). Deterministic failures, like malformed JSON or schema violation, are not retried: the failed finding is logged and not added to enrichment, the scan is marked partially enriched, and the original scanner finding is preserved. The three stage prompts and their response schemas are given in `docs/prompts appendix.md` in the released repository. SentinelAI performs no model training or fine-tuning; no weights are learned, and neither code nor data leaves the machine. Three generation requests are issued for each correlated group, and correlation, which is deterministic, completes before any model call. The synthesizer and verifier are deterministic modules that issue no generation requests.

4.3 Reporting, Container, and Continuous Integration

Five materializers are provided: terminal display, JSON, Markdown, HTML, and SARIF. The SARIF emitter was written from scratch for version 2.1.0 [9]. A distinct message type confines every model-derived property to the property bag, so the analyzer's message remains the analyzer's. The SARIF emitter maps each raw finding identifier to `partialFingerprints` and records the correlation

group identifier and the contributing scanners in the result properties. Ingestion into downstream code-scanning platforms has not been tested.

5 Evaluation Methodology and Engineering Validation

5.1 Scope and Non-Scope

The evaluation demonstrates observed behaviour that meets the intended model, the claimed invariants are true, and that the outputs are reproduced on the declared tool chain. It does not measure detection effectiveness. Precision, recall, F1, false positive rate, accuracy, expected calibration error, the reduction of alert fatigue, developer effort, the trust of the developer in the confidence value, is not reported here. No human-subject data or labeled positive and negative corpora were collected.

SentinelAI is not a detector: every finding comes from an established scanner, and the system's contribution is how findings are correlated, preserved, reported, and explained. We therefore compare it against the aggregation strategies it replaces (single scanners, an uncorrelated union, and naive deduplication) and ablate its own components (line tolerance, retrieval, the critic stage, and the verifier). An ML-detector baseline would measure detection accuracy, which requires a labeled corpus the paper explicitly does not claim.

5.2 Fixture, Environment, and Procedures

Measurements were collected on a deliberately small fixture of five Python files containing 94 lines of code (Table 4), held under version control so that the exact input can be replayed. The extended tier additionally scans a three-line `requirements.txt` pinning `flask 0.12.2`, `pyyaml 5.1`, and `requests 2.19.1`. Credential values in the fixture are deliberately synthetic and were chosen so that all relevant scanner rules still detect them.

Table 4: Benchmark fixture and seeded weaknesses.

File	LOC	Seeded weaknesses	CWE stated in code
`app/crypto utils.py`	6	MD5 password hashing	none
`app/db.py`	7	SQL built by string concatenation	none
`app/tasks.py`	29	`pickle.loads`, `os.system`, `subprocess` with `shell=True`, `yaml.load` without a loader, `eval`	none
`app/views.py`	37	reflected XSS, missing ownership check, SSRF, missing CSRF validation	79, 639, 918, 352
`config/settings.py`	15	two hardcoded credentials (synthetic)	none
Total	94		

Scanner versions are identical in the measurement environment and in the container image: Semgrep 1.172.0, Bandit 1.9.4, GitLeaks 8.30.1, Trivy 0.74.0, and OSV-Scanner 2.5.1. Measurements were taken on macOS 26.5.2 (Apple M5, 10 cores, 24 GB, ARM64) with Python 3.11.15 and Ollama 0.40.4, at commit `8d2a192`; continuous integration runs on Python 3.12.

6 Results and Discussion

Table 6 collects the readings.

Table 5: Validation methods.

Method	Measures	Repetitions
V1	Test suite, default and all markers	1 each
V2	Core tier: raw findings, groups, multi-scanner groups	5
V3	Extended tier: dependency findings per scanner	5
V4	Determinism of group membership and identifiers	5
V5	SARIF version, rule and result counts, schema validation	1
V6	Every exit code	1 each
V7	Enrichment verdicts and confidence	3

Table 6: Engineering validation and benchmark measurements.

Measurement	Observed value
Core tier, raw findings	23 (Semgrep 11, Bandit 11, GitLeaks 1)
correlated groups	16, of which 4 multi-scanner
Extended tier, raw / correlated	60 raw; 53 groups, 4 multi-scanner
Extended tier, dependency findings	37 (OSV-Scanner 25, Trivy 12); not reconciled
SARIF output (core fixture)	v2.1.0; 23 rules, 23 results; 33 unit tests + 10 schema-conformance tests
Enrichment / verifier (core fixture)	16 of 16 enriched; 9 verified, 7 rejected, 0 insufficient; identical in 3 of 3 runs
Confidence over the enriched set	mean 0.839, median 0.838, SD 0.023, range 0.795–0.882; identical in 3 of 3 runs
Exit codes	All five terminating conditions produced distinct exit codes
Core-tier runtime ($n = 5$)	mean 3.26 s (median 3.19 s, range 3.15–3.54 s)
Extended-tier runtime ($n = 5$)	mean 7.94 s
Enrichment runtime	mean 663 s for 48 generation calls (range 571–735 s; about 13.8 s per call)
Determinism check	Identical groups and identifiers on repeat runs

6.1 Discussion

On this fixture, the verifier rejected seven out of 16 enrichments, but this is not because of the quality of the explanation, it is because the scanner assigned a category label to the explanation. Each of the rejected groups is labeled with a generic name: Bandit's `blacklist` (4 groups) and `hashlib` (1 group) and Semgrep's `security` (2 groups). All groups having a descriptive label were checked. The rejected group contains one file like Bandit files `import subprocess` under `blacklist`, which is indeed the case, but the model explains the shell-injection risk, but it doesn't mention that word, so the response is rejected. The two `security` groups were confirmed only by the incidental presence of the word "security" in the model. The word-level fallback makes 14/16 tier decisions, while the word alone makes 2/16 tier decisions. The verifier therefore confirms that a response names the weakness in the scanner's own vocabulary; it does not discriminate between grounded and ungrounded explanations. A scanner-to-weakness vocabulary map would remove the label artifacts, and testing the check against adjudicated labels remains future work. Confidence behaves similarly: every finding has raw evidence, so the score reduces to (1+top retrieval score)/2, and its narrow range reflects similar retrieval scores across a homogeneous fixture rather than calibration.

6.2 Ablation and Baselines

Table 7 shows the correlation line tolerance for the 23 core raw findings and the relevance of the correlation to the aggregation strategies that it replaces.

Table 7: First: correlation line-tolerance ablation (core tier, 23 raw findings). Second: aggregation baselines (core tier).

Tolerance	Groups	Multi-scanner
0	17	3
1	16	4
2 (default)	16	4
3	16	4
5	15	4

Method	Shown	Raw not shown	Agreement visible
Bandit alone	11	12	No
Semgrep alone	11	12	No
GitLeaks alone	1	22	No
Union, no correlation	23	0	No
Naive dedup, one per (file, line)	15	8	No
Additive correlation (SentinelAI)	16	0	Yes

For tolerances 1-3, grouping is stable and the default grouping 2 is in the middle of this grouping plateau. On the other hand, when the SQL-injection pair is separated, e.g., at tolerance 0, Bandit will report that it cannot find the SQL query construction in line 6, but Semgrep will report that it cannot find the SQL execution in line 7. The two command-injection attempts found in the same file (on lines 17 and 21, both CWE-78) are merged at tolerance 5 into the same group, a false merge. The default is thus not optimised for the fixture.

Naive deduplication doesn't show any items more than correlation, but it drops evidence to do so: it drops 8 items including three that are actually identical across different scanners, replaces Semgrep's CWE-95 eval finding with Bandit's CWE-78 label for the same CWE, and hides a CWE-918 high severity SSRF finding behind a CWE-400 finding for missing timeout. Additive correlation involves a similar number of groups, but with each and every raw finding being retained as well as the scanners that match.

Table 8: Enrichment ablation on the 16 core groups. Replay rows re-score the saved responses of the full runs without new model calls.

Configuration	Verified	Rejected	Mean confidence	s per group
Full pipeline (3 runs)	9	7	0.839	41.5
No retrieval	11	5	0.500 (SD 0)	35.6
Verifier disabled	all 16 unverified		0.839	–
Phrase match only (replay)	2	14	–	–
Larger stop-word list (replay)	9	7	–	–

Removing retrieval sets confidence to 0.5 for each group, as the relevance is set to 0, but evidence strength remains 1, and two verdicts shift from rejected to verified through a wording drift, but not improved grounding – the verifier never. The critic feeds neither the verifier nor the confidence score, only the report's grounding verdict and its supported and unsupported claims, so removing it cannot change verdicts or confidence by construction and was not run separately. Restricting the verifier to whole-phrase matches rejects 14 of 16 groups, while widening the stop-word list changes nothing.

7 Limitations and Threats to Validity

The major setback is external validity. The reported measurements are based on one fixture of 5 Python files and 94 lines of code. The fixture is not representative of any real-world pipeline and therefore cannot be used to generalise to the production code base of different sizes, languages and dependency structures. The behaviour of the pipeline can be characterized only with a control group that has no artificial vulnerabilities, and quality metrics such as precision and recall can be computed only against a labeled corpus, which does not exist here. No calibration study was performed.

Generation uses temperature 0.0, but no seed is set, the model tags are not pinned by digest in the code, and identical text across runs or machines is therefore not guaranteed. The verifier is not verifying correctness but only grounding and its rejection verdicts have not been checked by humans.

There are a few gaps which have not been tackled. Semgrep runs with no explicit rule configuration, so its rule set is whatever the pinned version resolves by default; extended-tier results likewise depend on Trivy's and OSV-Scanner's vulnerability data at scan time, whose date is not recorded. The correlation rule admits two error modes that have not been quantified: two distinct weaknesses sharing a normalized CWE within two lines of one file would merge, and two reports of one weakness would stay separate if the analyzers disagree by more than two lines or normalize to different identifiers.

Grouping is transitive, so a chain of findings each within two lines of the next can merge findings that are further apart than the tolerance (for example, lines 10, 12, and 14 form one group). There are no CWE emitted by GitLeaks, so no multi-scanner group is added by it, and as a result, all multi-scanner groups on the fixture are contributed by Semgrep and Bandit. Only the first of a multi-CWE advisory is used. Positional identifiers (Bandit) and sequential group identifiers are reproducible for the same input, but may change if previous findings are removed or added.

8 Conclusion and Future Work

SentinelAI is a command-line DevSecOps framework that, according to its creators, adheres to the principle that "deterministic evidence" and "model interpretation" are two separate types of object and must not be confused. On a version-controlled fixture, reproducible correlation, structurally correct interchange output, fail-closed behaviour, and a grounding check that rejected 7 of 16 enrichments show that the check is reproducible but, on this fixture, tracks scanner vocabulary rather than explanation quality; quality metrics are not reported because the corpora needed to compute them were not assembled.

The open items include subprocess timeouts, commonizing Trivy and OSV-Scanner under a common vulnerability identity, commonizing the two stale benchmark assertions, and testing SARIF ingestion directly on platform backends such as GitHub Code Scanning.

References

[1] Imtiaz, N., Murphy, B., Williams, L.: How do developers act on static analysis alerts? An empirical study of Coverity usage. In: ISSRE, pp. 323–333. IEEE (2019)

[2] Bennett, G., Hall, T., Counsell, S., Winter, E., Shippey, T.: Do developers use static application security testing (SAST) tools straight out of the box? A large-scale empirical study. In: ESEM '24, pp. 454–460. ACM (2024)

[3] Hu, H., Wang, Y., Rubin, J., Pradel, M.: An empirical study of suppressed static analysis warnings. Proc. ACM Softw. Eng. **2**(FSE), 290–311 (2025) https://doi.org/10.1145/3715729

[4] Ullah, S., Han, M., Pujar, S., Pearce, H., Coskun, A., Stringhini, G.: LLMs cannot reliably identify and reason about security vulnerabilities (yet?): a comprehensive evaluation, framework, and benchmarks. In: IEEE S&P, pp. 862–880. IEEE (2024)

[5] Ding, Y., Fu, Y., Ibrahim, O., Sitawarin, C., Chen, X., Alomair, B., Wagner, D., Ray, B., Chen, Y.: Vulnerability detection with code language models: how far are we? In: ICSE, pp. 1729–1741. IEEE (2025)

[6] Risse, N., Böhme, M.: Uncovering the limits of machine learning for automatic vulnerability detection. In: 33rd USENIX Security Symposium, pp. 4247–4264 (2024)

[7] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-augmented generation for knowledge-intensive NLP tasks. In: NeurIPS, vol. 33, pp. 9459–9474 (2020)

[8] Du, X., Zheng, G., Wang, K., Zou, Y., Wang, Y., Deng, W., Feng, J., Liu, M., Chen, B., Peng, X., Ma, T., Lou, Y.: Vul-RAG: enhancing LLM-based vulnerability detection via knowledge-level RAG. ACM Trans. Softw. Eng. Methodol. **35**(10), 1–25 (2026) https://doi.org/10.1145/3797277

[9] Fanning, M.C., Golding, L.J. (eds.): Static Analysis Results Interchange Format (SARIF) Version 2.1.0. OASIS Standard (2020). https://docs.oasis-open.org/sarif/sarif/v2.1.0/sarif-v2.1.0.html

[10] MITRE Corporation: Common Weakness Enumeration (CWE) List. https://cwe.mitre.org/