

Do LLM Vulnerability-Detection Agents Reason Better? An Empirical Decomposition of Robustness, Resistance, and Grounding

Ziyu Chen* ✉ 

College of Electronic Engineering,
National University of Defense Technology,
Hefei, Anhui, China

Yuwei Li† ✉ 

College of Electronic Engineering,
National University of Defense Technology,
Hefei, Anhui, China

Shiwen Ou ✉ 

College of Electronic Engineering,
National University of Defense Technology,
Hefei, Anhui, China

Lu Zhang ✉ 

College of Electronic Engineering,
National University of Defense Technology,
Hefei, Anhui, China

Qiangpu Chen* ✉ 

College of Electronic Engineering,
National University of Defense Technology,
Hefei, Anhui, China

Taiyan Wang ✉ 

College of Electronic Engineering,
National University of Defense Technology,
Hefei, Anhui, China

Qingsong Xie ✉ 

College of Electronic Engineering,
National University of Defense Technology,
Hefei, Anhui, China

Zulie Pan† ✉ 

College of Electronic Engineering,
National University of Defense Technology,
Hefei, Anhui, China

Abstract

Background. Agents have recently improved LLM-based vulnerability detection through retrieval, iterative analysis, and verifier modules. However, existing evaluations mainly report improvements in detection performance without determining whether these improvements reflect more reliable vulnerability reasoning or access to retrieved context.

Aims. This study investigates how different agent components affect the reasoning-oriented evaluation dimensions of agent-based vulnerability detection.

Method. We present a controlled measurement framework with six conditions and evaluate it on six base models. The conditions progress from plain single-pass prompting through matched-context replay and ReAct-style tool use to self-review, trace-isolated verification, and multi-role decomposition, allowing us to separate context effects, iterative orchestration, and added agent modules under a shared protocol. Beyond vulnerability detection performance metrics, our framework evaluates three reasoning-oriented dimensions: *robustness* under semantics-preserving perturbations, *resistance* to misleading non-causal context, and *grounding* in concrete code evidence.

Results. The full agent often achieves higher accuracy or F1 than plain single-pass prompting, but these improvements do not consistently extend to the three measured behavioral dimensions of robustness, resistance, and grounding. The models are not consistently more stable under harmless code changes, are often misled by non-causal prompt cues, and rarely cite the exact code lines that explain the vulnerability. In the strongest improvement case, much of the accuracy change already appears when the model receives the same retrieved tool outputs without the full agent process; after that context is held fixed, the remaining agent effect is mainly associated with predicting vulnerable more often.

* co-first authors.

† Corresponding authors.



Conclusions. Higher detection performance should not be taken as direct evidence of better vulnerability reasoning. In our experiments, better performance metrics can reflect retrieved context or changes in how often the model predicts vulnerable, rather than a clear improvement in stable, evidence-based vulnerability reasoning. Future evaluations should report these behavior checks alongside performance metrics.

2012 ACM Subject Classification Security and privacy → Software security engineering; Computing methodologies → Artificial intelligence; Software and its engineering → Software testing and debugging

Keywords and phrases Large language models, vulnerability detection, software security, agent-based systems, empirical evaluation

Digital Object Identifier 10.4230/LIPIcs.ESEM.2026.28

Category Technical Track Paper

Supplementary Material *Collection (Replication package):* <https://github.com/cielverse/llm-vuln-agent-reasoning-eval>

1 Introduction

Vulnerability detection is a semantic judgment about code, not only a binary classification task. A detector must decide whether reachable behavior can violate a security property by following dependencies, recognizing guards or sanitizers, locating dangerous operations, and understanding patch effects. Classical work grounds such judgments in data/control flow, source-sink relations, and patch behavior [35, 13, 1]. Recent LLM evaluations show why this matters: models can appear accurate while relying on identifiers, formatting, surrounding snippets, or other shallow cues instead of the causal code path [29, 19, 25].

We use *reliable vulnerability reasoning* as an operational term grounded in observable detector behavior. A detector is more reliable when it bases its decisions on security-relevant semantics and causal evidence. Examples include data or control flows to dangerous operations, guards and sanitizers, causal reasoning, and patch effects, rather than surface cues or non-causal context. We operationalize this notion through three dimensions used throughout the paper: *robustness* to semantics-preserving code changes, *resistance* to misleading non-causal context, and *grounding* in concrete code evidence and consistent rationale. These dimensions follow prior evaluations that separate semantic code analysis from pattern matching [19] and check whether explanations match the true root cause [29].

LLM-based vulnerability detection has advanced rapidly. Agent detectors combine tools, retrieved context, iterative analysis, verification, and multi-role decomposition, and often report higher precision, recall, or F1 than plain prompting [37, 9, 20, 32, 34, 24, 28, 23]. However, a full-agent-versus-plain-baseline comparison changes several factors at once: the available information, how it is selected and processed, the organization of model calls, and potentially the predicted-label distribution. Existing studies usually compare complete systems or remove individual modules without holding the agent-selected tool observations fixed. This leaves a specific evaluation gap: aggregate improvements cannot be attributed to retrieved information, residual iterative processing over that information, or a shift toward one label. JITVUL’s report that some configurations predict “vulnerable” on more than 90% of inputs illustrates the risk: recall and F1 can rise without better pair-level discrimination [37]. Without such attribution, higher end-to-end metrics do not establish more reliable vulnerability reasoning or show whether added orchestration is warranted.

Research questions. Motivated by this gap, we ask: **RQ1.** How well do agent-based conditions preserve their verdicts under semantics-preserving repository changes? **RQ2.** How well do agent-based conditions resist non-causal repository distractors and opposite-label prompt cues? **RQ3.** How well are agent-based vulnerability predictions grounded in code evidence and consistent under paired patches? These questions operationalize robustness, resistance, and grounding, respectively.

Approach. To answer these questions, we combine a matched attribution chain with three behavioral evaluations. The chain compares plain single-pass prompting (C1), matched-context single-pass replay (L1), and the corresponding interactive ReAct agent (C2). L1 receives exactly the observations selected by the matched C2 agent. Because C1→C2 changes both the available context and the process used to select and analyze it, C1→L1 captures change recoverable from the C2-selected trace, whereas L1→C2 estimates the conditional residual of interactive execution under C2’s retrieval policy. C3–C5 test common extensions beyond the basic ReAct condition: self-review, trace-isolated verification, and plan–execute–verify role decomposition. They assess whether self-correction, independent re-analysis, or role specialization adds value beyond the basic agent condition [20, 32, 23].

We apply the six conditions to JITVUL across six base models. Repository-synchronized semantics-preserving perturbations address RQ1, misleading-context interventions address RQ2, and evidence localization, CWE classification, and paired-patch counterfactuals address RQ3. SecVulEval and SV-TrustEval-C provide complementary grounding and semantic-change checks. RQ1 and RQ2 are label-preserving, while the paired-patch component of RQ3 tests response to a causal label change.

Evaluation. We conduct a controlled, matched, cross-model evaluation. Within each contrast, the base model, analysis instance, repository snapshot, output schema, and execution controls are held fixed. This design combines aggregate detection metrics with controlled behavioral checks and tests whether the observed patterns recur across base models.

Applying this framework, we find that reported improvements in vulnerability detection should be interpreted cautiously. The central pattern is not that additional agent components reliably improve the measured reasoning-oriented dimensions, but that improvements in aggregate detection metrics are model-dependent and partly entangled with predicted-vulnerable-rate shifts. In the strongest aggregate-metric-improvement replication, DeepSeek-V3.2 improves by +0.279 F1, while pair-level accuracy changes by only +0.054; on the full matched sample, much of the C1→C2 pair-level accuracy change is already present after replaying the C2-selected tool trace, although the exact recovery ratio is stratum-sensitive. These improvements also do not consistently lift the reasoning-oriented dimensions: perturbation flip rates rise on five of six base models, self-review increases rather than reduces sensitivity, and grounding remains near the floor across conditions.

Contributions. This paper makes the following contributions:

- **Measurement Framework.** We define a reusable measurement framework that attributes observed changes to the portion recoverable through replaying an agent-selected tool trace and the residual associated with interactive execution over that trace. It organizes robustness, resistance, and grounding as complementary behavioral evaluation dimensions centered on a matched-context single-pass baseline. We will release the framework implementation and experimental artifacts as a public replication package.

- **Controlled Empirical Evaluation.** We apply the framework in a controlled cross-model study with six executable conditions and three intervention families. JITVUL serves as the main paired benchmark, while SecVulEval and SV-TrustEval-C provide the grounding and semantic-change checks used to answer RQ1–RQ3.
- **Empirical Analysis of Agent Gains.** Our experiments show that adding agent components does not consistently improve robustness, resistance, or grounding. Across six base models, reported gains co-vary with predicted-vulnerable-rate shifts, and matched-context replay reveals that a substantial portion of pair-level accuracy change already emerges after replaying the C2-selected tool trace.

2 Background and Related Work

2.1 Tool-Using LLM Agents

Agent-based vulnerability detection augments a target-function prompt with repository-level tool use. A ReAct-style agent alternates reasoning, tool calls, and observations before issuing a verdict [36]; tools may retrieve repository evidence such as definitions, dependencies, data-flow facts, or patch context [17, 37, 9]. We distinguish *retrieved context*, the evidence returned by tools, from *iterative reasoning*, the process that selects tools and updates the analysis. This distinction creates an attribution problem: improvements in aggregate detection metrics may come from the selected tool trace, the residual ReAct-loop effect, self-review or verification, or a predicted-vulnerable-rate shift. Section 3 therefore defines matched-context conditions that hold the C2-selected tool trace fixed.

2.2 Plain-LLM Vulnerability Detection

LLM-based vulnerability detection builds on earlier ML and deep-learning work on graph-based detectors, CVE-derived datasets, and vulnerability-detection baselines [39, 7, 3]. Recent plain-LLM work improves vulnerability detection through prompting, model adaptation, and context enhancement [10, 14, 27, 38, 6, 22, 16, 33]. These studies improve end-task performance, but a complementary diagnostic line asks what detectors actually rely on. Perturbation and semantic-reasoning benchmarks expose weak causal grounding; results vary sharply by model and prompt; and syntactic metrics can match or manipulate detector predictions [29, 19, 25, 21, 31]. These findings motivate tests of stability under semantics-preserving changes, resistance to non-causal context, and root-cause grounding.

2.3 Agent-Based Vulnerability Detection

Repository-level analysis has pushed vulnerability detection from single LLM calls to tool-using agents. JITVUL, RepoAudit, and recent multi-agent systems add ReAct loops, memory or validators, role decomposition, routing, parallel agents, or confidence-based escalation [37, 9, 32, 20, 34, 24, 28, 23]. Existing ablations are useful for testing whether a module matters, but they usually remove validators, rounds, or escalation stages from the full pipeline. They therefore do not separate the value of the selected tool trace from the residual value of iterating over that trace on the same input. Related agent benchmarks also show that aggregate performance can depend strongly on architecture or implementation choices [2, 18]. Table 1 summarizes these gaps and our coverage.

■ **Table 1** Evaluation-only comparison of prior work. Columns cover scope, attribution design, and robustness/reasoning checks. ✓ direct; ● partial; ○ limited; – not addressed. The shaded row marks our coverage; Faith./Counterf. apply when the pre-patch verdict is **vulnerable**.

Work	Scope			Attribution Design					Robustness and Reasoning						
	Subj.	Repo.	Agg.	X-model	Ctrl.	Ctx./orch.	Verif.	iso.	Repo-sync.	Aug.	Mislead.	Evi.	Root.	Faith.	Counterf.
Plain-LLM evaluations and diagnostics															
Steenhoek et al. [27]	LLMs	–	✓	✓	–	–	–	–	–	–	–	○	–	–	–
SecLLMHolmes [29]	LLMs	–	✓	✓	✓	–	–	–	–	✓	–	–	✓	✓	–
PrimeVul / How Far Are We? [5]	LLMs	–	✓	✓	✓	–	–	–	–	–	–	–	–	–	–
Lin & Mohaisen [21]	LLMs	–	✓	✓	–	–	–	–	–	–	–	–	–	–	–
SV-TrustEval-C [19]	LLMs	–	✓	✓	✓	–	–	–	–	✓	–	–	✓	–	–
Lost in the End [26]	LLMs	–	✓	✓	✓	–	–	–	–	●	–	✓	–	○	–
Weissberg et al. [31]	LLMs	–	✓	✓	✓	–	–	–	–	✓	–	–	–	–	–
Agent benchmarks and evaluations															
JITVUL [37]	Agents	✓	✓	✓	✓	●	–	–	–	–	–	–	○	○	–
SEC-bench [15]	Agents	✓	✓	✓	●	–	●	–	–	–	–	–	–	○	–
SastBench [8]	Agents	✓	✓	✓	–	●	–	–	–	–	–	–	–	–	–
This work	Agents	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Key abbreviations: Ctrl.=controlled comparison; Ctx./orch.=context-orchestration attribution; Verif. iso.=verification isolation; Repo-sync.=repository-synchronized perturbation.

3 Methodology

We use a paired, matched design in which each contrast fixes the analysis instance, repository snapshot, tool suite, and output schema while varying only the condition or intervention. L1 is the central matched-context control, and three intervention families evaluate robustness, resistance, and grounding.

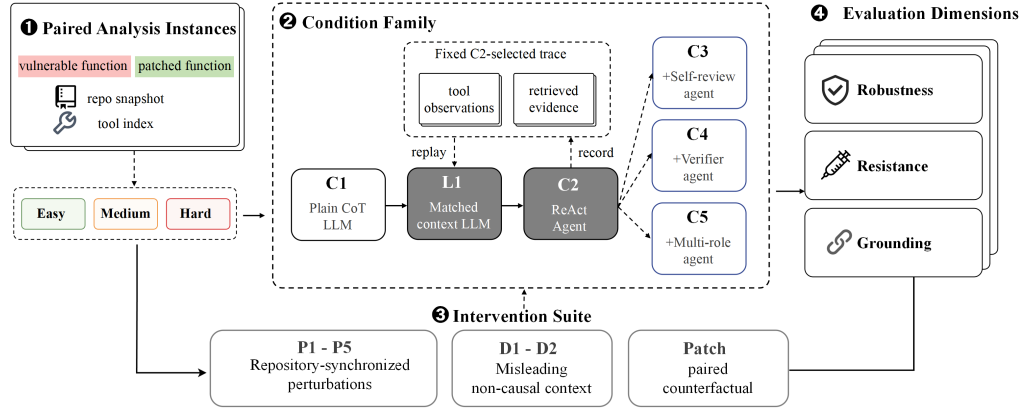
3.1 Conditions

We study six conditions that isolate design primitives commonly varied in current agent-based vulnerability detectors: tool access, iterative reasoning, self-review, independent verification, and multi-role decomposition. Figure 1 summarizes the paired instances, condition family, and interventions. The main attribution chain is C1→L1→C2. C3, C4, and C5 test additional agent components. We reserve C-labels for independently executable detector configurations; L1 denotes a matched trace-replay baseline because it is defined only after C2 has selected a tool trace.

C1 (plain CoT) sees only the target function and emits a single-pass verdict. **L1** receives exactly the tool observations that C2 retrieved on the same input, in the same order, and then emits a single-pass verdict. **C2** (ReAct agent) iterates through thought–action–observation cycles with dynamic tool calls. **C3** adds self-review on top of C2: the model inspects its own reasoning after the verdict and revises only if it identifies a concrete mistake. **C4** adds an external, trace-isolated verifier that sees neither the detector’s verdict nor its reasoning trace; the detector and verifier use the same output schema, and an evidence-aware merge rule produces the final verdict while preserving both intermediate records for audit. The merge keeps the detector verdict when detector and verifier agree; when they disagree, it selects the verdict whose record cites at least one target-file evidence line and has higher confidence, falling back to the verifier verdict if neither side provides target-file evidence. **C5** implements a plan–execute–verify multi-role architecture [20, 32, 23]. We treat C5 as a bundled-architecture comparison rather than a single ablation step.

The C1-vs-C2 contrast mixes two effects: the information contained in the C2-selected tool trace and the residual effect of the ReAct loop over that trace. L1 fixes the tool trace selected by C2, following prior work that treats access to retrieved information and reasoning over that information as distinct capabilities [12, 30]. The step C1→L1 measures how much of the C2 change is recoverable once the model is given the same tool trace selected by the agent. The L1→C2 step captures the remaining difference between static trace replay and interactive execution, including the tool-selection dynamics embedded in C2’s trajectory. The chain therefore provides an attribution under the agent’s retrieval policy: C1→L1 is the replay-recoverable component, and L1→C2 is the interactive residual. A fixed-policy retriever, such as call-graph retrieval, would provide a complementary attribution under a different selection policy (§ 6).

RQ1 uses C1→L1→C2 as its primary chain, with C3–C5 supplementary; RQ2 compares C1, C2, and C4-aware/blind under D1–D2; and RQ3 evaluates all six conditions, with patch consistency restricted to eligible reruns (Table 2).



■ **Figure 1** Study framework. Paired vulnerable–patched instances feed the condition family; P1–P5, D1–D2, and patch counterfactuals audit robustness, resistance, and grounding.

3.2 Interventions

The intervention families map directly to the RQs: RQ1 uses P1–P5 for robustness, RQ2 uses D1–D2 for resistance, and RQ3 combines paired-patch counterfactuals with evidence and CWE analyses for grounding. P1–P5 and D1–D2 preserve the ground-truth label and target reachability. Code-changing interventions rebuild prompts and tool indices; patch replacement is the only label-changing intervention and is evaluated separately for RQ3.

■ **Table 2** Research-question-to-design mapping.

RQ	Datasets	Conditions	Interv.	Measures	Analysis	Interp.
RQ1	JITVUL; SV-TrustEval-C (aux.)	C1→L1→C2; C3–C5	P1–P5	FR; Δ -Acc; PFC	matched; CIs	robustness
RQ2	JITVUL	C1; C2; C4 aware/blind	D1–D2	FR; cue response; reversals	McNemar; directional est.; corr.;	resistance
RQ3	JITVUL; SecVulEval	C1; L1; C2–C5	paired patch	CWE Top-1/3; evidence F1; Faith	audit	grounding

Repository perturbations (P1–P5). For RQ1, Table 3 summarizes five semantics-preserving changes: renaming identifiers (P1), changing function names (P2), adding unreachable code (P3), inserting dead functions (P4), and formatting changes (P5). These perturbations test *robustness*: whether detectors maintain consistent predictions under small, semantics-preserving code changes. Together they cover surface signals that vulnerability detectors may overuse: lexical cues in identifiers and function names, visible but non-causal code within or outside the target function, and presentation-level formatting. All edits run on temporary repository copies with the tool index rebuilt. P1 and P2 rename variables and functions consistently across prompt, repository, and call sites. P3 inserts syntactically valid `if (0)` blocks containing vulnerability-looking operations, P4 adds unused helper functions with vulnerability-like patterns, and P5 applies AST-preserving whitespace normalization. After each perturbation, definition lookup, caller/callee tracing, and file-level retrieval all operate on the same perturbed snapshot shown to the model. We validate perturbations by checking that the target function remains locatable, the repository index can be rebuilt, and the ground-truth vulnerable/safe label remains unchanged; failed materializations are excluded from the matched denominator for both members of the comparison.

Misleading non-causal context (D1–D2). For RQ2, Table 3 summarizes two interventions that test *resistance*. D1 injects unrelated repository code and rebuilds the corresponding tool index, so the distractor is available through the same retrieval interface as other repository content. The injected code is drawn from a different project or non-overlapping module, contains vulnerability-looking tokens or API calls, is not reachable from the target function, and is checked not to modify the target function, patch, call graph, or ground-truth label. D2 adds a one-sentence opposite-label prompt cue (vulnerable samples are framed as safe, and safe samples are framed as vulnerable). This is an adversarial stress test for non-causal task framing: realistic workflows may contain fallible cues such as issue reports, review comments, static-alert summaries, or commit messages, and D2 intentionally instantiates the worst-case opposite-label version of such a cue. The D2 cue is not evidence and does not modify the repository, patch, call graph, or label; it tests whether the model follows a non-causal prompt cue over code evidence. The RQ2 analysis focuses on C1, C2, and C4 because these conditions separate plain prompting, tool-using agent analysis, and independent verification. For C4, we distinguish *aware* verification (the verifier sees the misleading signal) from *blind* verification (the verifier sees only the clean target). This contrast tests whether verification helps through re-analysis or through controlled exposure.

Patch counterfactuals and grounding checks. For RQ3, this intervention tests *grounding* through the vulnerable–patched pair structure. Unlike P1–P5 and D1–D2, it is not label-preserving. For vulnerable samples, we replace the target function with its paired patched version and rebuild the corresponding prompt and tool index. We then evaluate grounding along three aspects. Evidence localization asks whether the cited code overlaps vulnerability-relevant code. CWE classification asks whether the predicted weakness type matches the ground truth. Patch-based counterfactual consistency asks whether a `vul` verdict changes to `safe` after the paired patch removes the vulnerability. Detailed evaluation procedures are described in § 4.4. Table 3 summarizes the three intervention families.

■ **Table 3** Intervention taxonomy.

Dim.	ID	Change and signal tested
Rob.	P1	Rename local identifiers; tests variable/parameter lexical cues.
	P2	Rename target functions and call sites; tests function-symbol lexical cues.
	P3	Add an unreachable block inside the target; tests local, visible but non-executable code.
	P4	Add an unused helper outside the target; tests repository-level code off the call path.
	P5	Reformat code without AST changes; tests presentation-only cues.
Res.	D1	Inject unrelated repository code; tests retrievable but non-causal code evidence.
	D2	Add an opposite-label prompt cue; tests prompt cues without code evidence.
Grd.	Patch	Replace vulnerable target with paired patch; label-changing causal counterfactual.

■ **Table 4** Model pool.

Model	Params	Access	Spec.	Role
Claude Sonnet 4.6	–	Closed	Gen.	Strat.
gpt-5.4	–	Closed	Gen.	Eval.
DeepSeek-V3.2	671B	Open	Gen.	Eval.
gpt-oss-120b	120B	Open	Gen.	Eval.
Qwen3.6-27B	27B	Open	Gen.	Eval.
DeepSeek-Coder-V2-Lite	16B	Open	Code	Eval.
Qwen2.5-Coder-14B	14B	Open	Code	Eval.

4 Experimental Setup

4.1 Research Questions

As introduced in Section 1, the study is organized around three research questions corresponding to robustness, resistance, and grounding. This section operationalizes them through the interventions and condition comparisons defined in Section 3.

RQ1. How well do agent-based conditions preserve their verdicts under semantics-preserving repository changes? This RQ focuses on understanding whether agent-based vulnerability detectors are robust to semantics-preserving code perturbations. We examine P1–P5 perturbations and use the $C1 \rightarrow L1 \rightarrow C2$ attribution chain to distinguish robustness changes already present after replaying the C2-selected tool trace from residual changes introduced by the ReAct loop. We also use an external semantic-change check to ensure that stability under semantics-preserving changes does not simply reflect insensitivity to real semantic changes. The semantic-change check is auxiliary rather than part of the matched JITVUL perturbation experiment.

RQ2. How well do agent-based conditions resist non-causal repository distractors and opposite-label prompt cues? In this RQ, we examine whether tool-using agents and verifier-based conditions can ignore misleading non-causal context that is visible to the model but not related to the vulnerability. We analyze repository-level distractors, opposite-label prompt cues, and the aware/blind verifier contrast for C4 to separate independent verification from controlled exposure to the misleading signal.

RQ3. How well are agent-based vulnerability predictions grounded in code evidence and consistent under paired patches? With this RQ, we examine whether verdict-level gains are accompanied by stronger observable grounding in vulnerability evidence. We evaluate grounding through evidence localization, CWE classification, and patch-based counterfactual consistency when vulnerable code is replaced by its paired patch.

4.2 Datasets and Sampling

Dataset roles. We use three datasets with distinct experimental roles. JITVUL is the main dataset for all RQs because it provides repository-level context, vulnerable–patched function pairs, and CWE annotations. This structure supports perturbation tests, misleading non-causal context tests, and patch-based counterfactuals on the same paired units. SecVulEval is used for the RQ3 evidence-localization analysis because it provides statement-level evidence labels, which allow us to evaluate whether cited evidence overlaps with vulnerability-relevant code. SV-TrustEval-C is used as an external semantic-change check for RQ1. It is not merged into the matched JITVUL perturbation experiment; instead, it checks whether conditions that appear stable under semantics-preserving perturbations still respond to genuine semantic changes.

Difficulty-stratified paired sampling. Our main sample uses difficulty-stratified paired sampling over eligible JITVUL pairs with complete repository snapshots. For each pair p , we compute a pair-level competence score $s(p) \in \{0, \dots, 6\}$ by counting correct zero-shot CoT predictions across the vulnerable and patched versions under three reference models: gpt-5.4, Claude Sonnet 4.6, and DeepSeek-V3.2. We partition pairs into $\mathcal{S}_{\text{easy}} = \{s \in \{4, 5\}\}$, $\mathcal{S}_{\text{med}} = \{s = 3\}$, and $\mathcal{S}_{\text{hard}} = \{s \in \{0, 1, 2\}\}$. We then draw 50 pairs from each stratum. The resulting JITVUL sample contains 150 pairs, or 300 function instances. These pairs define the statistical units of the main study. Each pair is reused across conditions, perturbation states, misleading-context variants, base models, and patch-based counterfactual reruns. The experiment therefore estimates behavioral changes within pairs.

Rationale. The paired design requires a detector to distinguish a vulnerable function from its patched counterpart rather than rely on project-level, style-level, or vulnerability-prior shortcuts. Stratification serves a separate purpose. It ensures that the experiment covers cases that current LLMs usually solve, cases on which they disagree, and cases that all reference models find difficult. This prevents the mechanism analysis from being dominated by the natural difficulty distribution of the dataset.

Supplemental stratified evidence set. For SecVulEval, we construct a difficulty-stratified supplemental sample with 50 instances per stratum. This set is used only for the RQ3 evidence-localization analysis, where statement-level labels are needed to measure evidence localization directly. JITVUL remains the primary dataset for pair-level verdicts, CWE evaluation, and patch-based counterfactuals.

Voting/evaluation overlap. gpt-5.4 and DeepSeek-V3.2 appear in both the voting pool and the evaluation pool. We therefore interpret difficulty-stratified results through within-model condition contrasts. The voting procedure defines strata before the agent conditions are compared; it does not affect the matched within-model contrasts reported here.

4.3 Models and Execution Pipeline

Model selection. We selected six base models to span three dimensions relevant to agent-based vulnerability detection: model access, scale, and code specialization. The pool includes closed-weight frontier models, large open-weight general-purpose models, and smaller code-specialized models. Open-weight models range from 14B to 671B parameters, covering small,

mid, and large tiers. One additional closed-weight frontier model is used only for difficulty stratification (Table 4). We treat a finding as cross-model only when its direction holds for a majority of the six base models.

Execution controls. All matched conditions use the same repository snapshot, tool suite, output schema, deterministic decoding setting ($T=0$), fixed per-sample seed, and versioned repository index. The tool suite includes target-function lookup, caller/callee tracing, symbol search, file-content retrieval, and static-alert retrieval when available. Each tool-using role runs under the same per-role tool-call budget; if the budget is exhausted, that role must produce its output from the observations already collected. Multi-role conditions can therefore cost more overall, but no individual role receives a larger retrieval budget. Prompt templates and tool descriptions are identical across base models, with no model-specific tuning. These controls keep the comparison focused on condition design rather than differences in infrastructure or prompting.

Output schema and parsing. Every condition is parsed into the same structured record: `verdict` in `{vulnerable, safe}`, `numeric confidence`, `evidence_lines`, `evidence_files`, `evidence_functions`, ranked `cwe_top3`, and a free-text rationale. Agent conditions also store the full thought-action-observation trace and tool errors; C4 additionally stores the detector and verifier records before the merge. We score only parsed records with a valid binary verdict. Data errors such as a missing target function or failed snapshot materialization are removed pairwise from the affected matched comparison rather than imputed as wrong predictions, so condition denominators can differ slightly; each table reports them.

4.4 Evaluation Protocol

Experimental procedure. The main JITVUL experiment uses 150 vulnerable-patched pairs (300 function instances) as the statistical units. Each instance is evaluated under the six conditions using the same repository snapshot, tool suite, and output schema. We then reuse the same paired units for matched intervention runs: P1-P5 for RQ1, D1-D2 for RQ2, and patch-based counterfactual reruns for RQ3. Supplemental SecVulEval and SV-TrustEval-C runs are evaluated separately for statement-level evidence and semantic-change checks. Intervention runs add observations, not independent samples.

Matched analysis. All primary comparisons are within-model contrasts on matched samples. For detection performance, we report instance-level metrics and pair-level accuracy (pAcc), where a pair is correct only if both the vulnerable and patched versions are classified correctly. For condition effects, we compare predictions made by the same base model on the same sample and snapshot. We report both pooled results over the 150 JITVUL pairs and stratified results by difficulty. Cross-model claims are based on whether the direction of the within-model effect holds for a majority of base models.

Measures and statistical support. Clean detection uses Acc, Precision, Recall, F1, MCC, pAcc, and VulBias. VulBias is the predicted-vulnerable rate, i.e., the fraction of valid instances predicted `vulnerable`; pAcc is pair-level accuracy and is correct only when both members of a vulnerable-patched pair receive the expected labels. RQ1 reports flip rate (FR), accuracy change (Δ -Acc), and pairwise flip consistency (PFC). RQ2 reports FR, susceptibility to the injected cue, and D2 reversal direction, including clean-correct cases destroyed, denoted

CC→WH, and clean-wrong cases repaired, denoted CW→Corr. RQ3 reports CWE Top-1/Top-3 accuracy, evidence F1 against SecVulEval statement labels, and patch-replacement consistency (Faith), following rationale/evidence and faithfulness evaluation work [4, 11]. Faith is an auxiliary subset-dependent diagnostic computed on eligible vulnerable instances that are initially predicted **vulnerable**.

For matched binary verdicts, we use McNemar’s exact test; for other metrics, we report bootstrap confidence intervals. Reported p -values support planned matched contrasts, not a multiple-comparison discovery procedure. Cross-cell patterns are interpreted through effect sizes, available intervals, and directional consistency across matched units. With $N = 150$ pairs, we treat small effects as interval estimates rather than definitive null results.

5 Experimental Results

This section first calibrate aggregate performance and then evaluate RQ1–RQ3. Unless stated otherwise, claims are within-model matched contrasts; cross-model statements require a recurring direction across the six complete JITVUL pipelines.

5.1 Baseline Calibration

Result. The C2 ReAct agent reproduces improvements in aggregate detection metrics on several base models, but the same C1→C2 transition shows heterogeneous patterns: clear positive pair-level accuracy change on some base models, near-neutral change on others, and change dominated by a shift in the predicted-vulnerable rate on others. These aggregate improvements therefore need to be examined alongside robustness, resistance, and grounding before they can support claims about reasoning reliability. The conditional L1 contrast estimates the residual ReAct-loop effect over a fixed C2-selected trace.

The clearest replication is DeepSeek-V3.2. F1 increases from 0.351 to 0.630, while pAcc moves only from 0.068 to 0.122. The matched-context replay condition changes the interpretation: on the full matched sample, replaying the C2-selected observations in L1 reproduces much of the observed C1→C2 pAcc change, while the remaining L1→C2 pAcc change is only +0.007 and is not detected by the matched test at this scale, with $p = 1.0$. C2’s larger visible effect is instead a predicted-vulnerable-rate shift, with DeepSeek’s VulBias moving from 0.270 at C1 to 0.805 at C2.

The other base models show why the calibration matters. The same agent scaffold improves pair-level detection for Qwen3.6-27B and Qwen2.5-Coder-14B, leaves it approximately unchanged for gpt-5.4 and DS-Coder-V2-Lite, and degrades it through a safe-side collapse for gpt-oss-120b. C5 follows the same model-dependent pattern, so role decomposition alone does not explain the aggregate gains.

The main C1→C2 direction persists across strata, but L1’s exact recovery ratio is stratum-dependent; aggregate metrics alone therefore do not establish better robustness, resistance, or grounding.

5.2 RQ1: Robustness to Semantics-Preserving Perturbations

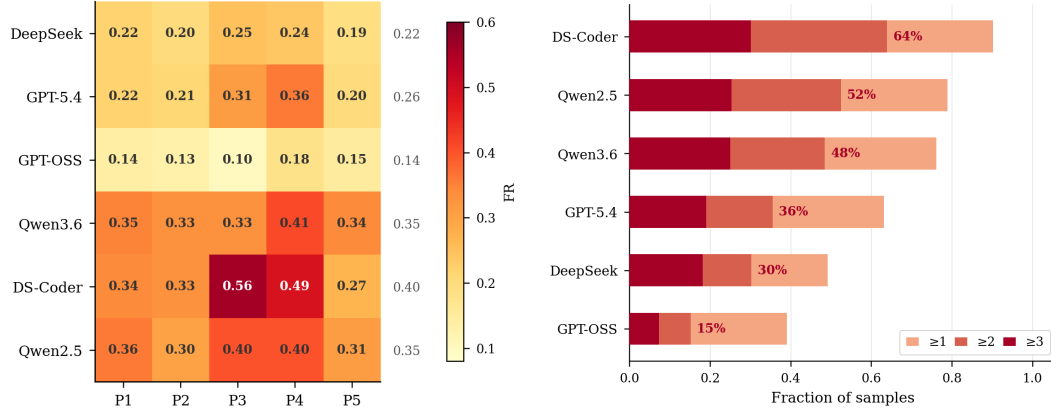
Result. The condition chain does not make verdicts more stable under semantics-preserving edits in our matched perturbation runs. From C1 to C2, aggregate flip rate increases on five of six complete pipelines. The exception is gpt-oss-120b, whose lower FR co-occurs with its collapse toward **safe**, not with better discrimination. The important pattern is repeated reversibility: for the four most affected base models, more than half of evaluable samples flip under at least two of the five semantics-preserving perturbation families.

■ **Table 5** Clean JITVUL performance, pooled across difficulty strata. $\Delta p\text{Acc}/F1/\text{VulBias}$ are computed against the same model’s C1. $\uparrow/\downarrow$ mark pAcc change against the preceding step in $C1 \rightarrow L1 \rightarrow C2$. **L1** is the matched-context baseline. C2 shading marks clear pair-level gain sage, strong predicted-vulnerable-rate shift or collapse coral, and near-neutral change gray. C5 is a bundled multi-role reference condition, not part of the $C1 \rightarrow L1 \rightarrow C2$ ablation chain.

Model	Cond.	Absolute metric (clean baseline)					Delta vs. C1		
		Acc	F1	MCC	pAcc	VulBias	$\Delta p\text{Acc}$	$\Delta F1$	ΔVB
DeepSeek-V3.2	C1	0.500	0.351	+0.000	0.068	0.270	—	—	—
	L1	0.522	0.445	+0.045	0.116	0.364	+0.048 $\uparrow$	+0.094	+0.094
	C2	0.519	0.630	+0.049	0.122	0.805	+0.054 $\uparrow$	+0.279	+0.535
	C3	0.510	0.580	+0.021	0.144	0.666	+0.076	+0.229	+0.396
	C4	0.535	0.642	+0.091	0.156	0.801	+0.088	+0.291	+0.531
	C5	0.542	0.628	+0.097	0.163	0.734	+0.095	+0.277	+0.464
gpt-5.4	C1	0.563	0.627	+0.135	0.207	0.670	—	—	—
	L1	0.577	0.634	+0.162	0.189	0.654	−0.018 $\downarrow$	+0.007	−0.016
	C2	0.574	0.609	+0.150	0.223	0.591	+0.016 $\approx$	−0.018	−0.079
	C3	0.570	0.592	+0.142	0.216	0.554	+0.009	−0.035	−0.116
	C4	0.584	0.617	+0.170	0.209	0.587	+0.002	−0.010	−0.083
	C5	0.581	0.587	+0.161	0.230	0.517	+0.023	−0.040	−0.153
gpt-oss-120b	C1	0.577	0.506	+0.160	0.234	0.357	—	—	—
	L1	0.535	0.444	+0.076	0.177	0.333	−0.057 $\downarrow$	−0.062	−0.024
	C2	0.532	0.187	+0.128	0.102	0.074	−0.132 $\downarrow$	−0.319	−0.283
	C3	0.512	0.321	+0.031	0.172	0.217	−0.062	−0.185	−0.140
	C4	0.535	0.312	+0.101	0.189	0.171	−0.045	−0.194	−0.186
	C5	0.534	0.255	+0.112	0.128	0.122	−0.106	−0.251	−0.235
Qwen3.6-27B	C1	0.535	0.276	+0.065	0.107	0.159	—	—	—
	L1	0.543	0.374	+0.126	0.110	0.215	+0.003 $\approx$	+0.098	+0.056
	C2	0.566	0.535	+0.134	0.252	0.434	+0.145 $\uparrow$	+0.259	+0.275
	C3	0.564	0.461	+0.146	0.191	0.301	+0.084	+0.185	+0.142
	C4	0.588	0.569	+0.179	0.292	0.448	+0.185	+0.293	+0.289
	C5	0.525	0.434	+0.046	0.165	0.350	+0.058	+0.158	+0.191
DS-Coder-V2-Lite	C1	0.502	0.372	+0.002	0.111	0.295	—	—	—
	L1	0.515	0.251	+0.039	0.069	0.149	−0.042 $\downarrow$	−0.121	−0.146
	C2	0.493	0.330	−0.015	0.144	0.257	+0.033 $\approx$	−0.042	−0.038
	C3	0.536	0.427	+0.075	0.200	0.312	+0.089	+0.055	+0.017
	C4	0.531	0.424	+0.061	0.207	0.321	+0.096	+0.052	+0.026
	C5	0.490	0.428	−0.021	0.144	0.392	+0.033	+0.056	+0.097
Qwen2.5-Coder-14B	C1	0.527	0.385	+0.059	0.073	0.271	—	—	—
	L1	0.500	0.403	+0.000	0.030	0.338	−0.043 $\downarrow$	+0.018	+0.067
	C2	0.539	0.547	+0.078	0.178	0.518	+0.105 $\uparrow$	+0.162	+0.247
	C3	0.546	0.554	+0.091	0.178	0.514	+0.105	+0.169	+0.243
	C4	0.545	0.548	+0.090	0.183	0.509	+0.110	+0.163	+0.238
	C5	0.485	0.407	−0.035	0.126	0.376	+0.053	+0.022	+0.105

The instability follows the same mechanism exposed by the clean baseline. When the agent changes the predicted-vulnerable rate, a small non-semantic edit can be enough to move a borderline trace to the other side of the decision boundary. DeepSeek increases from 0.157 FR at C1 to 0.219 at C2 with $p < 0.001$, and gpt-5.4 increases from 0.199 to 0.259 with $p < 0.001$. Qwen3.6-27B, DS-Coder-V2-Lite, and Qwen2.5-Coder-14B reach still higher C2 FR levels of 0.353, 0.398, and 0.355, respectively. The accuracy change under these perturbations is small or negative, so the flips are not mainly benign corrections. These flips indicate unstable verdicts under label-preserving repository changes.

The perturbation profile explains why this happens. P3 and P4 insert vulnerability-looking but non-causal code; P5 only changes presentation. Yet at C2, P5 alone flips roughly one fifth to one third of decisions on four base models, and P3/P4 are frequently the strongest perturbations for the code-specialized models. In the raw traces, flipped cases often show the agent reusing the same tools but shifting the rationale from the target data flow to a newly salient surface cue, such as an unused helper or an unreachable block. The added context thus appears to provide more non-causal anchors for the answer.



■ **Figure 2** RQ1 robustness under repository-synchronized semantics-preserving perturbations. The left panel reports C2 flip rates by perturbation family P1–P5, and the right panel reports the concentration of repeated flips under C2. The low FR for gpt-oss-120b should be read together with its safe-collapse in Table 5.

■ **Table 6** Flip rate stratified by ground-truth label, averaged over P1–P5 and pooled across difficulty strata. N is the number of matched instances per GT class; Δ FR is relative to the same base model’s C1 row. Bold cells mark the largest C1→C2 jump in both GT classes.

Cond.	Model	N		FR		Δ FR		Model	N		FR		Δ FR	
		vul	safe	vul	safe	vul	safe		vul	safe	vul	safe	vul	safe
C1	DeepSeek	148	148	0.159	0.154	–	–	Qwen3.6	119	127	0.284	0.296	–	–
L1		148	149	0.189	0.184	+0.030	+0.030		108	102	0.294	0.256	+0.010	–0.040
C2		148	149	0.196	0.242	+0.037	+0.088		121	124	0.373	0.334	+0.089	+0.038
C1	gpt-5.4	150	150	0.171	0.227	–	–	DS-Coder	134	135	0.245	0.243	–	–
L1		149	149	0.200	0.256	+0.029	+0.029		144	145	0.294	0.292	+0.049	+0.049
C2		149	149	0.247	0.271	+0.076	+0.044		145	146	0.402	0.393	+0.157	+0.150
C1	gpt-oss	144	145	0.291	0.267	–	–	Qwen2.5	137	139	0.296	0.296	–	–
L1		148	147	0.267	0.274	–0.024	+0.007		135	135	0.325	0.309	+0.029	+0.013
C2		148	147	0.163	0.117	–0.128	–0.150		135	135	0.352	0.358	+0.056	+0.062

Self-review does not consistently reduce this failure mode. C3 raises FR above C2 on five of six base models rather than dampening it. That pattern is consistent with self-review reopening the same unstable boundary instead of adding independent evidence. SV-TrustEval-C provides a semantic-change sanity check: the condition-level changes are small, mostly within about three points, so the RQ1 result is specific to repository-scale semantics-preserving edits that introduce plausible but non-causal security cues.

Ground-truth-stratified flip rate. To separate instability from shifts in the predicted-vulnerable rate, we also stratify perturbation flips by the ground-truth label of the perturbed instance. Each JITVUL pair contributes one vulnerable instance and one patched-safe instance, giving a balanced, exogenous conditioning variable for each base-model–condition cell. Table 6 reports pooled P1–P5 flip rates for the GT-vulnerable and GT-safe subsets.

This control strengthens the robustness result. On five of six base models, C1→C2 FR increases in both GT classes simultaneously. The increases are +0.037/+0.088 for DeepSeek-V3.2 and +0.157/+0.150 for DS-Coder-V2-Lite on GT-vulnerable and GT-safe instances. A pure predicted-vulnerable-rate shift would tend to move the two class-conditional flip rates in opposite directions. Here, they rise together. The exception, gpt-oss-120b, nearly always predicts **safe** at C2, leaving few flip opportunities in either class.

Finding 1: Agent verdicts become easier to flip without semantic change.

Semantics-preserving edits raise flip rates on most base models, and the increase appears in both vulnerable and safe classes on the same five of six base models. Within this model pool, the agent loop is therefore associated with additional boundary instability beyond a shifted predicted-vulnerable rate.

5.3 RQ2: Resistance to Misleading Non-Causal Context

Result. In the adversarial opposite-label prompt-cue stress test, agents are more vulnerable to prompt-level misleading context than to repository distractors. At C2, D2 has a higher flip rate than D1 on every non-collapsed base model, and it does so in the most damaging direction: correct verdicts are much easier to destroy than wrong verdicts are to repair under this cue design.

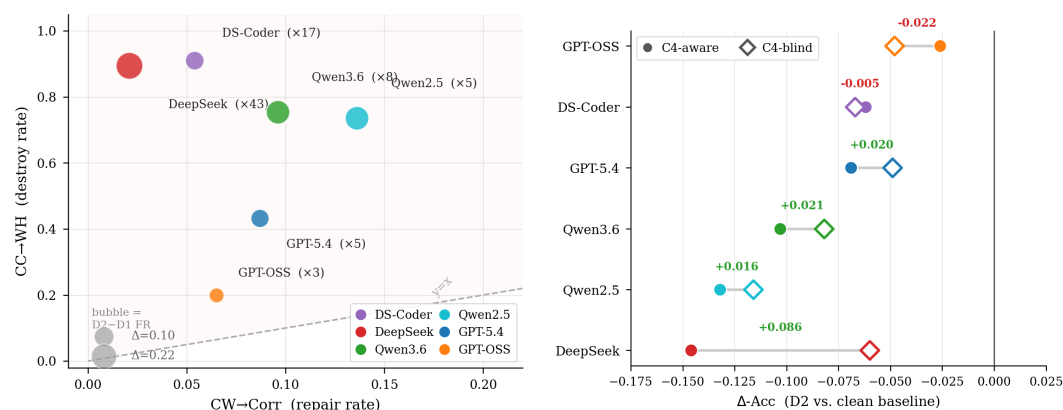


Figure 3 RQ2 misleading-context effects. Under D2 at C2, the left panel shows that the destruction of clean-correct verdicts greatly exceeds the repair of clean-wrong verdicts. The right panel compares C4-aware and C4-blind accuracy loss under D2; blind verification reduces the damage by isolating the verifier from the cue.

The asymmetry is large enough to be visible without relying on aggregate accuracy. Under D2 at C2, the share of clean-correct verdicts turned wrong is 0.895 for DeepSeek-V3.2, 0.754 for Qwen3.6-27B, 0.736 for Qwen2.5-Coder-14B, and 0.910 for DS-Coder-V2-Lite; the corresponding repair rates for clean-wrong verdicts are only 0.021, 0.096, 0.136, and 0.054. The opposite-label prompt cue therefore does not simply add noise. It creates a directed reversal in this stress test: the cue can override code evidence when the model is right but rarely repairs a wrong verdict.

The mechanism is exposure order in this stress test. D2 frames the analysis before tool observations are interpreted, whereas D1 must first be retrieved and connected to the target. D2 therefore produces larger FR than D1 in the matched DeepSeek and gpt-5.4 tests at C2, with $p < 0.001$ and $p = 0.015$; gpt-oss-120b again collapses toward **safe**.

The C4 verifier separates two possible explanations. If verification helped because a second model pass reasons more carefully, then an aware verifier that sees the same D2 cue should recover. It does not. The recovery appears when the verifier is blind to the cue. On DeepSeek-V3.2, the D2 flip rate drops from 0.454 with aware verification to 0.302 with blind verification, with $p < 0.001$; the accuracy loss is reduced from -0.411 to -0.251 . The same direction holds on five of six base models. Stratified by difficulty, the DeepSeek

blind-minus-aware accuracy gap under D2 grows from +0.105 on Easy to +0.173 on Medium and +0.202 on Hard. The hardest cases benefit most because they are precisely where the initial agent is most susceptible to using the prompt cue as a tie-breaker.

Finding 2: Opposite-label prompt cues create directed reversals; blind verification limits them. Prompt-level opposite-label cues mostly destroy clean-correct verdicts rather than repair clean-wrong ones. Verification helps only when the verifier is cue-blind, implicating controlled exposure rather than an additional reasoning round.

5.4 RQ3: Grounding of Vulnerability Predictions

Result. No condition provides consistent evidence that the aggregate performance gain lifts grounding above the low baseline observed in these metrics. CWE classification remains low, statement-level evidence F1 stays near the floor on SecVulEval, and patch-replacement consistency is decoupled from the C1→C2 pAcc change. The agent often produces more elaborate explanations, but the additional explanation surface is not accompanied by reliably better localization of the causal vulnerable lines.

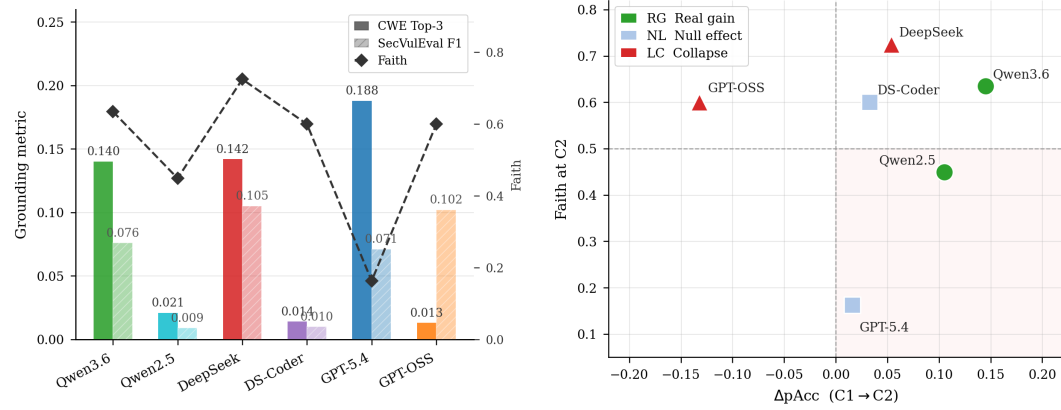


Figure 4 RQ3 grounding diagnostics. The left panel reports C2 CWE Top-3, best SecVulEval statement-level evidence F1, and C2 Faith. The right panel compares C1→C2 pAcc change with C2 Faith. Detection change and grounding diagnostics are not aligned across base models.

At C2, CWE Top-3 accuracy is at most 0.188 and is near zero for the two smaller code models. It does not track pAcc: gpt-5.4 falls from 0.287 at C1 to 0.188 at C2, while Qwen2.5-Coder-14B remains at 0.021 despite a pAcc gain. On SecVulEval, no complete pipeline (C2–C5) exceeds evidence F1 0.105; the decline in DeepSeek precision from L1 to C4 is consistent with over-citation rather than better causal localization.

Statement-level evidence on SecVulEval gives a stricter test of this observable grounding signal. Among complete agent pipelines (C2–C5), no condition exceeds evidence F1 0.105 in the main evidence evaluation. DeepSeek reaches the strongest complete-pipeline score at C2, but precision falls from L1 to C4 from 0.081 to 0.029, with C2 and C3 at 0.074 and 0.060. The trace pattern is consistent with over-citation: iterative passes cite more plausible lines, but these additional citations do not consistently become more causal.

■ **Table 7** SecVulEval soft-overlap F1 on vulnerable samples with non-empty evidence predictions, pooled across difficulty strata. $F1_{\pm k}$ matches citations within $\pm k$ lines of the labeled lines; N is the number of evaluated samples.

Model	Cond.	N	± 0	± 1	± 3	± 5	Model	Cond.	N	± 0	± 1	± 3	± 5
DeepSeek	L1	34	0.087	0.227	0.304	0.374	Qwen3.6	L1	32	0.055	0.156	0.268	0.298
	C2	35	0.099	0.187	0.250	0.295		C2	29	0.069	0.153	0.187	0.211
	C3	35	0.093	0.172	0.232	0.252		C3	30	0.050	0.177	0.233	0.263
	C4	36	0.086	0.174	0.236	0.278		C4	31	0.030	0.093	0.127	0.208
	C5	36	0.088	0.171	0.254	0.299		C5	32	0.045	0.108	0.184	0.205
gpt-5.4	L1	36	0.073	0.179	0.246	0.265	DS-Coder	L1	35	0.001	0.002	0.005	0.007
	C2	36	0.061	0.120	0.172	0.221		C2	19	0.005	0.010	0.017	0.023
	C3	36	0.035	0.089	0.119	0.150		C3	13	0.003	0.006	0.013	0.018
	C4	35	0.049	0.137	0.171	0.212		C4	18	0.009	0.014	0.023	0.031
	C5	35	0.055	0.142	0.177	0.216		C5	21	0.004	0.007	0.013	0.018
gpt-oss	L1	35	0.109	0.232	0.293	0.376	Qwen2.5	L1	25	0.000	0.000	0.000	0.000
	C2	33	0.086	0.162	0.226	0.282		C2	16	0.000	0.000	0.000	0.000
	C3	31	0.090	0.180	0.248	0.297		C3	16	0.000	0.000	0.000	0.000
	C4	22	0.098	0.198	0.277	0.312		C4	14	0.001	0.002	0.005	0.007
	C5	21	0.022	0.089	0.106	0.166		C5	16	0.001	0.001	0.003	0.004

Soft-overlap and audit. Because strict evidence F1 can miss adjacent citations, we also report soft overlap within $\pm k$ lines, for $k \in \{0, 1, 3, 5\}$, and complement it with a manual audit. Table 7 shows that at ± 3 , F1 rises by about 2.5–4 $\times$, but all scores remain below 0.40 even at ± 5 . L1 matches or exceeds C2 at ± 3 for DeepSeek-V3.2, gpt-5.4, and gpt-oss-120b, while the two smaller code models remain near zero.

The 20-trace inspection attributes the remaining gap to three patterns: *off-by-block citation*, where the cited block ends next to the labeled bug; *over-citation around the bug*, where the agent cites a wide block containing the label but with low precision; and *evidence/verdict mismatch*, where the cited lines overlap the bug but the final verdict is still **safe**. Thus, relaxed matching shows that strict evidence F1 understates some near misses, but it does not overturn the grounding result.

Exploratory causal-grounding audit on 50 traces. Soft overlap measures proximity, not whether the rationale identifies the causal vulnerability mechanism. We therefore manually rated 50 vulnerable C2 traces from the four base models with non-trivial evidence emission, stratified by Easy, Medium, and Hard. The sample contains 13 traces each from DeepSeek-V3.2, gpt-5.4, and gpt-oss-120b, plus 11 traces from Qwen3.6-27B. A trace is *causal* if the cited evidence reaches the labeled line or its ± 1 neighborhood and the rationale matches the true mechanism; *adjacent* if it localizes nearby code or a related mechanism; and *irrelevant* otherwise. One author rated each trace using the cited lines, SecVulEval **changed_lines**, rationale text, and CWE predictions. We use the resulting audit as qualitative support for interpreting the quantitative grounding analyses.

Table 8 summarizes the exploratory sample: 15 of 50 traces (30%) were rated causal, 19 (38%) adjacent, and 16 (32%) irrelevant. The descriptive causal shares range from 27% to 31% across the four represented models. The audited cases also illustrate a divergence between verdict correctness and grounding: 8 of 21 correct vulnerable predictions were rated causal, while 7 of 29 missed bugs still cited the bug lines.

Three rated cases illustrate the audit categories. A causal DeepSeek-V3.2 trace on CVE-2017-14064 (function `cState_space_set` in the Ruby `json` extension) identifies the allocator/deallocator mismatch even though one endpoint is adjacent rather than exactly cited. An adjacent trace on CVE-2019-19275 (function `ast2obj_slice` in `typed_ast`) cites a 20-line

■ **Table 8** Exploratory manual three-way grounding rating on 50 vulnerable C2 traces. Right blocks split by predicted verdict; all audited inputs are GT-vulnerable, so “pred. vul.” is a correct verdict and “pred. safe” is a missed bug. C/A/I denotes causal, adjacent, irrelevant.

Model	N	All audited			Pred. vul., correct				Pred. safe, missed			
		C	A	I	n	C	A	I	n	C	A	I
DeepSeek-V3.2	13	4	7	2	9	3	5	1	4	1	2	1
gpt-5.4	13	4	4	5	2	1	0	1	11	3	4	4
gpt-oss-120b	13	4	5	4	4	2	1	1	9	2	4	3
Qwen3.6-27B	11	3	3	5	6	2	2	2	5	1	1	3
Total	50	15	19	16	21	8	8	5	29	7	11	11
Share	–	30%	38%	32%	–	38%	38%	24%	–	24%	38%	38%

block around the bug, recovering the neighborhood but not isolating the causal statements. An irrelevant gpt-oss-120b trace on CVE-2015-2778 (function `CoreUserInputHandler::handleSay` in `quassel`) cites header lines and a distant helper but misses the actual bug.

Read with the soft-overlap table, the manual rating provides qualitative context for the grounding result. The audited cases illustrate how strict evidence F1 can miss nearby evidence and how longer traces may cite plausible but non-causal code. The quantitative grounding conclusion rests on the full evidence-localization and patch-based analyses.

Denominators behind the grounding metrics. Faith and evidence F1 are subset-dependent. At C2, empty `evidence_lines` rates range from 0.7% to 60.6% across base models, the soft-overlap evidence denominators range from 117 to 296 emitted-evidence instances, and the patch-replacement-eligible subsets range from 10 to 120 instances. We therefore treat patch-replacement consistency as an auxiliary diagnostic rather than a basis for ranking base models; its cross-model association with C1→C2 pAcc gain is near zero. The descriptive correlation is $r = +0.10$ over the six base models.

Finding 3: Longer traces do not lift grounding above the observed floor.

Relaxed line matching shows that strict evidence F1 understates some near misses, while the exploratory manual ratings illustrate causal, adjacent, and irrelevant evidence patterns. Across the quantitative diagnostics, the full agent does not consistently lift grounding signals above the matched-context baseline, and patch-replacement consistency does not track clean pair-level gain.

Cost note. Table 9 reports the full cost and latency results. On DeepSeek-V3.2, C2 and C4 cost about 9× and 15× C1, respectively, while much of the C1→C2 pAcc change is already present in L1. Thus, the generic ReAct loop is costly for its residual clean-task benefit; blind verification is the clearer exception because it limits cue exposure.

6 Discussion

Lessons learned. Three lessons emerge: aggregate gains require attribution because trace replay and prediction-rate shifts can explain much of the change; additional orchestration does not consistently improve robustness, resistance, or grounding; and under D2, blind verification helps by controlling cue exposure.

Research implications. Researchers and benchmark designers should pair aggregate metrics with matched trace-replay controls, pair-level accuracy, predicted-vulnerable rate, and behavioral probes. Reporting eligible denominators alongside replay-recoverable and residual effects makes gains easier to interpret.

Practical implications. Tool builders should require measured behavioral gains to justify agent cost; keep retrieved evidence, verifier exposure, and decision aggregation auditable; and validate cue-isolated verification on natural inputs before deployment.

Threats to validity. *Construct validity.* Robustness, resistance, and grounding are operational probes, and L1 effects are conditional on the C2-selected trace. *Internal validity.* Prompt wording, tool behavior, provider variability, and implementation may affect outcomes despite matched controls; overlapping difficulty-voting and evaluation models restrict strata to within-model contrasts. *External validity.* JITVUL chiefly supports paired repository-level detection, the evaluated conditions do not cover all agent architectures, and weaker natural cues may differ from D2. *Conclusion validity.* Evidence F1 and Faith use varying eligible subsets. The 50-trace, one-author audit is qualitative; conclusions rely on matched effects, confidence intervals, and cross-model consistency.

7 Conclusion

This paper presented a matched measurement framework for auditing LLM vulnerability-detection agents beyond aggregate accuracy. By holding the C2-selected tool trace fixed, the framework separates change recoverable from replaying the selected tool trace from the residual ReAct-loop effect over that same context, and evaluates the same conditions on robustness, resistance, and grounding.

Across six base models, aggregate agent gains reproduce in some settings but do not consistently imply better measured vulnerability reasoning. The clearest positive lesson is controlled independence: blind verification can reduce susceptibility to misleading prompt cues. Future vulnerability-detection agents should therefore demonstrate gains on robustness, resistance, and grounding, not only on aggregate detection metrics.

Data Availability Statement

The replication package is publicly available at <https://github.com/cielverse/llm-vuln-agent-reasoning-eval>. It contains the implementation, data, prompts, model-output records, and reproduction instructions. API keys and provider-side logs are excluded; runs that depend on external services must be regenerated following the documented procedures.

References

- 1 Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In *Proc. PLDI*, pages 259–269, 2014. doi:10.1145/2594291.2594299.
- 2 Tara Bogavelli, Roshnee Sharma, and Hari Subramani. AgentArch: A comprehensive benchmark to evaluate agent architectures in enterprise. *arXiv:2509.10769*, 2025.
- 3 Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. Deep learning based vulnerability detection: Are we there yet? *IEEE Transactions on Software Engineering*, 48(9):3280–3296, 2022. doi:10.1109/TSE.2021.3087402.

- 4 Jay DeYoung, Sarthak Jain, Nazneen Fatema Rajani, Eric Lehman, Caiming Xiong, Richard Socher, and Byron C. Wallace. ERASER: A benchmark to evaluate rationalized NLP models. In *Proc. ACL*, pages 4443–4458, 2020. doi:10.18653/V1/2020.ACL-MAIN.408.
- 5 Yangruibo Ding, Yanjun Fu, Omniyyah Ibrahim, Chawin Sitawarin, Xinyun Chen, Basel Alomair, David A. Wagner, Baishakhi Ray, and Yizheng Chen. Vulnerability detection with code language models: How far are we? In *Proc. ICSE*, pages 1729–1741, 2025. doi:10.1109/icse55347.2025.00038.
- 6 Xiaohu Du, Ming Wen, Jiahao Zhu, Zifan Xie, Bin Ji, Huijun Liu, Xuanhua Shi, and Hai Jin. Generalization-enhanced code vulnerability detection via multi-task instruction fine-tuning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10507–10521, 2024. doi:10.18653/v1/2024.findings-acl.625.
- 7 Jiahao Fan, Yi Li, Shaohua Wang, and Tien N. Nguyen. A C/C++ code vulnerability dataset with code changes and CVE summaries. In *Proc. MSR*, pages 508–512, 2020. doi:10.1145/3379597.3387501.
- 8 Jake Feiglin and Guy Dar. SastBench: A benchmark for testing agentic SAST triage. *arXiv:2601.02941*, 2026. doi:10.48550/arXiv.2601.02941.
- 9 Jinyao Guo, Chengpeng Wang, Xiangzhe Xu, Zian Su, and Xiangyu Zhang. RepoAudit: An autonomous LLM-agent for repository-level code auditing. In *Proc. ICML*, volume 267 of *Proceedings of Machine Learning Research*, pages 21083–21100, 2025. URL: <https://openreview.net/forum?id=TXcifVbFpG>.
- 10 Larry Huynh, Yinghao Zhang, Djimon Jayasundera, Woojin Jeon, Hyounghick Kim, Tingting Bi, and Jin B. Hong. Detecting code vulnerabilities using LLMs. In *Proc. DSN*, pages 401–414, 2025. doi:10.1109/dsn64029.2025.00047.
- 11 Alon Jacovi and Yoav Goldberg. Towards faithfully interpretable NLP systems: How should we define and evaluate faithfulness? In *Proc. ACL*, pages 4198–4205, 2020. doi:10.18653/V1/2020.ACL-MAIN.386.
- 12 Mingyu Jin, Weidi Luo, Sitao Cheng, Xinyi Wang, Wenyue Hua, Ruixiang Tang, William Yang Wang, and Yongfeng Zhang. Disentangling memory and reasoning ability in large language models. In *Proc. ACL*, pages 1681–1701, 2025. doi:10.18653/V1/2025.ACL-LONG.84.
- 13 Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. Static analysis for detecting taint-style vulnerabilities in web applications. *Journal of Computer Security*, 18(5):861–907, 2010. doi:10.3233/JCS-2009-0385.
- 14 Avishree Khare, Saikat Dutta, Ziyang Li, Alaia Solko-Breslin, Rajeev Alur, and Mayur Naik. Understanding the effectiveness of large language models in detecting security vulnerabilities. *arXiv:2311.16169*, 2023. doi:10.1109/icst62969.2025.10988968.
- 15 Hwiwon Lee, Ziqi Zhang, Hanxiao Lu, and Lingming Zhang. SEC-bench: Automated benchmarking of LLM agents on real-world software security tasks. In *Proc. NeurIPS*, 2025. URL: <https://openreview.net/forum?id=QQhQIqons0>.
- 16 Ahmed Lekssays, Hamza Mouhcine, Khang Tran, Ting Yu, and Issa Khalil. LLMxCPG: Context-aware vulnerability detection through code property graph-guided large language models. In *Proc. USENIX Security Symposium*, pages 489–507, 2025. URL: <https://www.usenix.org/conference/usenixsecurity25/presentation/lekssays>.
- 17 Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proc. NeurIPS*, pages 9459–9474, 2020. URL: <https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html>.
- 18 Han Li, Letian Zhu, Bohan Zhang, Rili Feng, Jiaming Wang, Yue Pan, Earl T. Barr, Federica Sarro, Zhaoyang Chu, and He Ye. Contextbench: A benchmark for context retrieval in coding agents. *arXiv*, 2026. arXiv:2602.05892.

- 19 Yansong Li, Paula Branco, Alexander M. Hoole, Manish Marwah, Hari Manassery Koduvely, Guy-Vincent Jourdan, and Stephan Jou. SV-TrustEval-C: Evaluating structure and semantic reasoning in large language models for source code vulnerability analysis. In *Proc. IEEE S&P*, pages 3014–3032, 2025. doi:10.1109/sp61157.2025.00191.
- 20 Youpeng Li, Kartik Joshi, Xinda Wang, and Eric Wong. MAVUL: Multi-agent vulnerability detection via contextual reasoning and interactive refinement. In *Proc. TPS-ISA*, pages 314–324, 2025. doi:10.1109/tps-isa67132.2025.00040.
- 21 Jie Lin and David Mohaisen. From large to mammoth: A comparative evaluation of large language models in vulnerability detection. In *Proc. NDSS*, 2025. doi:10.14722/ndss.2025.241491.
- 22 Fang Liu, Simiao Liu, Yinghao Zhu, Xiaoli Lian, and Li Zhang. Securereviewer: Enhancing large language models for secure code review through secure-aware fine-tuning. In *Proc. ICSE*, 2026. doi:10.48550/arXiv.2510.26457.
- 23 Renwei Meng, Haoyi Wu, Jingming Wang, and Haoyan Bai. VulnAgent-X: A layered agentic framework for repository-level vulnerability detection. *arXiv:2603.13384*, 2026. doi:10.48550/arXiv.2603.13384.
- 24 Shreshth Rajan. MultiVer: Zero-shot multi-agent vulnerability detection. *arXiv:2602.17875*, 2026. doi:10.48550/arXiv.2602.17875.
- 25 Niklas Risse, Jing Liu, and Marcel Böhme. Top score on the wrong exam: On benchmarking in machine learning for vulnerability detection. *Proc. ACM Softw. Eng.*, 2(ISSTA):388–410, 2025. doi:10.1145/3728887.
- 26 Francesco Sovrano, Adam Bauer, and Alberto Bacchelli. Large language models for in-file vulnerability localization can be “lost in the end”. *Proc. ACM Softw. Eng.*, 2(FSE):891–913, 2025. doi:10.1145/3715758.
- 27 Benjamin Steenhoek, Md Mahbubur Rahman, Monoshi Kumar Roy, Mirza Sanjida Alam, Earl T. Barr, and Wei Le. A comprehensive study of the capabilities of large language models for vulnerability detection. *CoRR*, abs/2403.17218, 2024. doi:10.48550/arXiv.2403.17218.
- 28 Chung-Nan Tsai, Xin Wang, Cheng-Hsiung Lee, and Ching-Sheng Lin. A sequential multi-stage approach for code vulnerability detection via confidence- and collaboration-based decision making. In *Proc. EMNLP*, pages 21151–21157, 2025. doi:10.18653/v1/2025.emnlp-main.1071.
- 29 Saad Ullah, Mingji Han, Saurabh Pujar, Hammond Pearce, Ayse K. Coskun, and Gianluca Stringhini. LLMs cannot reliably identify and reason about security vulnerabilities (yet?): A comprehensive evaluation, framework, and benchmarks. In *Proc. IEEE S&P*, pages 862–880, 2024. doi:10.1109/sp54263.2024.00210.
- 30 Mudit Verma, Siddhant Bhambri, and Subbarao Kambhampati. On the brittle foundations of ReAct prompting for agentic large language models. *CoRR*, abs/2405.13966, 2024. doi:10.48550/arXiv.2405.13966.
- 31 Felix Weissberg, Lukas Pirch, Erik Imgrund, Jonas Möller, Thorsten Eisenhofer, and Konrad Rieck. LLM-based vulnerability discovery through the lens of code metrics. In *Proc. ICSE*, 2026. doi:10.48550/arXiv.2509.19117.
- 32 Ratnadira Widayarsi, Martin Weyssow, Ivana Clairine Irsan, Han Wei Ang, Frank Liauw, Eng Lieh Ouh, Lwin Khin Shar, Hong Jin Kang, and David Lo. Let the trial begin: A mock-court approach to vulnerability detection using LLM-based agents. *arXiv:2505.10961*, 2025. doi:10.48550/arXiv.2505.10961.
- 33 Bozhi Wu, Chengjie Liu, Zhiming Li, Yushi Cao, Jun Sun, and Shang-Wei Lin. Enhancing vulnerability detection via inter-procedural semantic completion. *Proc. ACM Softw. Eng.*, 2(ISSTA):825–847, 2025. doi:10.1145/3728912.
- 34 Zihan Wu, Jie Xu, Yun Peng, Chun Yong Chong, and Xiaohua Jia. MulVul: Retrieval-augmented multi-agent code vulnerability detection via cross-model prompt evolution. *arXiv:2601.18847*, 2026. doi:10.18653/v1/2026.ac1-long.391.

- 35 Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. Modeling and discovering vulnerabilities with code property graphs. In *Proc. IEEE S&P*, pages 590–604, 2014. doi:10.1109/SP.2014.44.
- 36 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *Proc. ICLR*, 2023. doi:10.48550/arXiv.2210.03629.
- 37 Alperen Yildiz, Sin G. Teo, Yiling Lou, Yebo Feng, Chong Wang, and Dinil Mon Divakaran. Benchmarking LLMs and LLM-based agents in practical vulnerability detection for code repositories. In *Proc. ACL*, pages 30848–30865, 2025. doi:10.18653/v1/2025.acl-long.1490.
- 38 Xin Zhou, Ting Zhang, and David Lo. Large language model for vulnerability detection: Emerging results and future directions. In *Proc. ICSE*, pages 47–51, 2024. doi:10.1145/3639476.3639762.
- 39 Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In *Proc. NeurIPS*, pages 10197–10207, 2019. URL: <https://proceedings.neurips.cc/paper/2019/hash/49265d2447bc3bbfe9e76306ce40a31f-Abstract.html>.

A Cost and Latency

■ **Table 9** Per-condition inference cost per sample on RQ3 clean runs.

Cond.	DeepSeek			gpt-5.4			gpt-oss		
	Cost	Lat.	× C1	Cost	Lat.	× C1	Cost	Lat.	× C1
C1	0.003	12.4	1.0×	0.031	51.9	1.0×	–	19.8	1.0×
L1	0.002	6.7	0.7×	0.019	67.4	0.6×	–	13.8	0.7×
C2	0.026	97.4	9×	0.113	178.9	3.7×	–	31.1	1.6×
C3	0.027	104.7	9×	0.128	223.6	4.1×	–	46.7	2.4×
C4	0.044	179.9	15×	0.161	312.9	5.2×	–	71.0	3.6×
C5 [‡]	0.034	156.1	11×	0.157	331.0	5.1×	–	80.5	4.1×
Cond.	Qwen3.6			DS-Coder			Qwen2.5		
	Cost	Lat.	× C1	Cost	Lat.	× C1	Cost	Lat.	× C1
C1	–	92.0	1.0×	–	170.3	1.0×	–	41.1	1.0×
L1	–	41.5	0.5×	–	64.0	0.4×	–	14.2	0.3×
C2	–	569.7	6.2×	–	144.8	0.9×	–	125.9	3.1×
C3	–	554.2	6.0×	–	258.3	1.5×	–	156.9	3.8×
C4	–	793.7	8.6×	–	254.3	1.5×	–	193.6	4.7×
C5 [‡]	–	604.1	6.6×	–	382.9	2.2×	–	246.1	6.0×

Stratum-averaged values; cost in USD and latency in seconds. Shaded L1 rows cover replay only; local-model multipliers use latency. [‡]C5 is a bundled architecture reference, not part of the ablation chain.