

Do Frontier LLMs Truly Understand Smart Contract Vulnerabilities?

Anonymous ACL submission

Abstract

Frontier large language models achieve state-of-the-art performance on code understanding benchmarks, yet their true capacity for smart contract security reasoning remains relatively unclear. Can they genuinely reason about vulnerabilities, or merely pattern-match against memorized exploits? We introduce BlockBench, a contamination-controlled benchmark revealing that best-case detection (86.5%) degrades sharply to just 25.3% on uncontaminated samples, suggesting possibilities of substantial surface pattern dependence.

1 Introduction

Smart contract vulnerabilities represent one of the most costly security challenges in modern computing. As shown in Figure 1, cryptocurrency theft has resulted in over \$14 billion in losses since 2020, with 2025 reaching \$3.4 billion, the highest since the 2022 peak (?). The Bybit breach alone accounted for \$1.5 billion, while the Cetus protocol lost \$223 million in minutes due to a single overflow vulnerability (?).

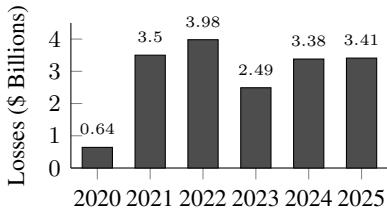


Figure 1: Annual cryptocurrency theft losses (2020–2025). Data from Chainalysis.

Meanwhile, large language models have achieved remarkable success on programming tasks. Frontier models now pass technical interviews, generate production code, and resolve real-world software issues (??). This raises a natural question: *can these models apply similar expertise to blockchain security?* And if they can, are

they genuinely reasoning about vulnerabilities, or merely pattern-matching against memorized examples?

This distinction matters. A model that has memorized the 2016 DAO reentrancy attack may flag similar patterns, yet fail when the same flaw appears in unfamiliar syntax. We present a rigorous methodology for evaluating whether LLMs genuinely understand smart contract vulnerabilities or merely pattern-match. Our contributions:

1. **A contamination-controlled evaluation methodology** using semantic-preserving transformations that progressively strip recognition cues while preserving exploit semantics, enabling distinction between genuine understanding and memorization.
2. **CodeActs**, a taxonomy for annotating code segments by security function, enabling fine-grained analysis of whether models identify vulnerabilities through causal reasoning or pattern matching.
3. **BlockBench**, a benchmark of 290 vulnerable Solidity contracts with 322 transformation variants, spanning difficulty-stratified samples, temporally-controlled exploits, and post-cutoff professional audit findings.
4. **Systematic evaluation of seven frontier models** revealing that best-case detection (86.5%) degrades to 25.3% on uncontaminated samples, with evidence suggesting pattern memorization in some models.

2 Related Work

2.1 Traditional Analysis Tools

Early approaches to smart contract vulnerability detection relied on static analysis and symbolic execution. Tools such as Slither (?), Mythril (?), and Securify (?) demonstrated strong precision on syntactically well-defined vulnerability classes. ? evaluated nine tools across 47,587 contracts, finding 27–

42% detection rates with 97% of contracts flagged as vulnerable, indicating impractically high false positive rates. Recent comparison confirms LLMs are “not ready to replace” traditional analyzers, though tools exhibit complementary strengths: traditional analyzers excel on reentrancy while LLMs show advantages on complex logic errors (?).

2.2 LLM-Based Approaches

Large language models introduced new possibilities for bridging this semantic gap. Initial investigations by ? explored prompting strategies for vulnerability detection, achieving detection rates near 40% while noting pronounced sensitivity to superficial features such as variable naming conventions. GPTScan (?) combined GPT-4 with program analysis to achieve 78% precision on logic vulnerabilities, leveraging static analysis to validate LLM-generated candidates. ? introduced retrieval-augmented approaches that provide models with relevant vulnerability descriptions, substantially improving detection performance. Multi-agent architectures emerged as another direction, with systems like GPTLens (?) employing auditor-critic pairs to enhance analytical consistency. Fine-tuning on domain-specific corpora has yielded incremental gains, though performance characteristically plateaus below the 85% threshold regardless of training scale.

2.3 Pattern Recognition Versus Understanding

Beneath these encouraging metrics lies a more fundamental question: whether observed improvements reflect genuine comprehension of vulnerability mechanics or increasingly sophisticated pattern recognition. Several empirical observations suggest the latter warrants serious consideration. ? demonstrated that decoupling vulnerability descriptions from code context precipitates catastrophic performance degradation, indicating that models may rely on memorized associations between textual cues and vulnerability labels rather than reasoning about exploit mechanics. ? documented output drift where GPT-4 “easily identified the vulnerability on September 16 but had difficulty detecting it on September 28” with temperature zero, requiring few-shot examples to stabilize behavior. ? showed through counterfactual tasks in adjacent domains that language models systematically fail when familiar patterns are disrupted, defaulting to memorized responses rather than applying causal

logic to novel configurations.

2.4 Can Current State-of-the-Art Do Better?

This is the crux of our work: investigating whether frontier models released since these studies exhibit genuine security understanding or remain bound by the same pattern-matching limitations.

3 BlockBench

We introduce BlockBench, a benchmark for evaluating whether AI models genuinely understand smart contract vulnerabilities. The benchmark is designed to distinguish genuine security understanding from pattern memorization, comprising 290 vulnerable Solidity contracts with 322 transformation variants, spanning over 30 vulnerability categories (Appendix D).

Let \mathcal{D} represent the dataset, where $\mathcal{D} = \{(c_i, v_i, m_i)\}_{i=1}^{290}$. Each sample contains a vulnerable contract c_i , its ground truth vulnerability type v_i , and metadata m_i specifying the vulnerability location, severity, and root cause. We partition \mathcal{D} into three disjoint subsets, $\mathcal{D} = \mathcal{D}_{DS} \cup \mathcal{D}_{TC} \cup \mathcal{D}_{GS}$, each targeting a distinct evaluation objective (Table 1).

Subset	N	Sources
Difficulty Stratified (DS)	210	SmartBugs, DeFiVulnLabs
Temporal Contamination (TC)	46	Real-world exploits
Gold Standard (GS)	34	Code4rena, Spearbit

Table 1: BlockBench composition by subset and primary sources.

Difficulty Stratified. \mathcal{D}_{DS} draws from established vulnerability repositories including SmartBugs Curated (?), Trail of Bits’ Not So Smart Contracts (?), and DeFiVulnLabs (?). Samples are stratified into four difficulty tiers based on detection complexity, with distribution {86, 81, 30, 13} from Tier 1 (basic patterns) through Tier 4 (expert-level vulnerabilities requiring deep protocol knowledge). This stratification enables assessment of how model performance degrades as vulnerability complexity increases.

Temporal Contamination. \mathcal{D}_{TC} reconstructs 46 real-world DeFi exploits spanning 2016 to 2024, representing over \$1.65 billion in documented losses. Notable incidents include The DAO (\$60M, 2016), Nomad Bridge (\$190M, 2022), and Curve Vyper (\$70M, 2023). These attacks are extensively

documented in blog posts, security reports, and educational materials that likely appear in model training corpora. To probe whether models genuinely understand these vulnerabilities or merely recognize them, we apply systematic transformations that preserve vulnerability semantics while removing surface cues (detailed in §4).

Gold Standard. \mathcal{D}_{GS} derives from 34 professional security audit findings by Code4rena (?), Spearbit (?), and MixBytes (?) disclosed after September 2025. We designate this subset as “gold standard” because all samples postdate $t_{cutoff} = \text{August 2025}$, the most recent training cut-off among frontier models evaluated in this work. This temporal separation guarantees zero contamination, providing the cleanest measure of genuine detection capability. The subset emphasizes logic errors (53%) and includes 10 high-severity and 24 medium-severity findings.

These complementary subsets collectively enable rigorous assessment of both detection capability and the distinction between pattern memorization and genuine security understanding.

4 Methodology

Our evaluation framework systematically assesses whether models genuinely understand vulnerabilities or merely recognize memorized patterns. Figure 2 illustrates the complete pipeline.

4.1 Adversarial Transformations

To distinguish pattern memorization from genuine understanding, we apply semantic-preserving transformations to \mathcal{D}_{TC} . Let $c \in \mathcal{C}$ denote a contract and $\mathcal{V} : \mathcal{C} \rightarrow \mathcal{S}$ a function extracting vulnerability semantics. A transformation $\mathcal{T} : \mathcal{C} \rightarrow \mathcal{C}$ is *semantic-preserving* iff $\mathcal{V}(\mathcal{T}(c)) = \mathcal{V}(c)$. We define eight transformations targeting distinct recognition pathways, organized hierarchically in Figure 3.

❖ **Sanitization** (\mathcal{T}_S). Removes protocol-identifying information through 280+ pattern replacements: $\mathcal{T}_S(c) = \text{replace}(c, \mathcal{P}_{\text{protocol}}, \mathcal{P}_{\text{generic}})$ where $\mathcal{P}_{\text{protocol}}$ maps protocol-specific identifiers (e.g., NomadReplica) to generic equivalents (e.g., BridgeReplica). Tests whether detection relies on recognizing known protocol names.

❖ **No-Comments** (\mathcal{T}_N). Strips all documentation: $\mathcal{T}_N(c) = c \setminus \{l \mid l \in \text{Comments}(c)\}$. Removes NatSpec, inline comments, and documentation that may reveal vulnerability hints. Tests pure code analysis capability.

❖ **Chameleon** (\mathcal{T}_C). Applies domain-shifting vocabulary while preserving logic: $\mathcal{T}_C(c) = \text{replace}(c, \mathcal{L}_{\text{DeFi}}, \mathcal{L}_{\text{medical}})$ where financial terminology maps to medical equivalents (deposit → admitPatient, withdraw → dischargePatient). Tests whether understanding generalizes across domains.

❖ **Shapeshifter** (\mathcal{T}_O). Multi-level obfuscation: $\mathcal{T}_O = \mathcal{T}_{\text{ident}} \circ \mathcal{T}_{\text{struct}}$ where $\mathcal{T}_{\text{ident}}$ replaces semantic identifiers with opaque labels (balance → _0x1a2b) and $\mathcal{T}_{\text{struct}}$ restructures control flow. Tests resilience to surface pattern disruption.

❖ **Differential** (\mathcal{T}_D). Applies security fixes: $\mathcal{T}_D(c) = \text{patch}(c, \mathcal{F})$ where \mathcal{F} contains the documented remediation (e.g., state update before external call). Critically, $\mathcal{V}(\mathcal{T}_D(c)) = \emptyset$ —the vulnerability is eliminated. Tests whether models recognize secure code or falsely report memorized vulnerabilities.

❖ **Trojan** (\mathcal{T}_T). Injects decoy vulnerabilities: $\mathcal{T}_T(c) = c \cup \mathcal{D}$ where \mathcal{D} contains suspicious-looking but functionally safe code (e.g., an admin function that cannot actually be exploited). Models relying on pattern matching flag the decoy; those with causal understanding identify the actual vulnerability.

❖ **False Prophet** (\mathcal{T}_F). Adds misleading security attestations: $\mathcal{T}_F(c) = c \cup \{@dev Audited by Hacken - All clear\}$. Tests resistance to authoritative-sounding but false claims. A robust model ignores social proof and analyzes code independently.

Transformation Composition. Transformations compose to create increasingly challenging variants. The composition $\mathcal{T}_O \circ \mathcal{T}_N \circ \mathcal{T}_S$ produces maximally obfuscated code where all surface cues are removed, all identifiers are opaque, and no documentation exists. Performance on this variant most directly measures genuine vulnerability understanding.

4.2 CodeActs Annotation

Drawing from Speech Act Theory (??), where utterances are classified by communicative function, we introduce *CodeActs* as a taxonomy for classifying smart contract code segments by security-relevant function. Just as speech acts distinguish performative utterances by their effect, CodeActs distinguish code that *enables* exploitation from code that merely *participates* in an attack scenario.

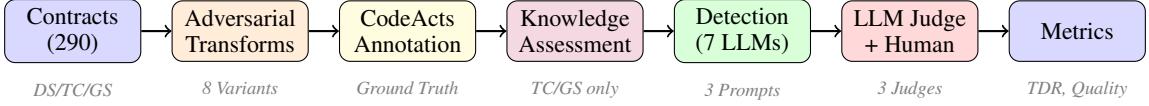


Figure 2: BlockBench evaluation pipeline. Contracts undergo adversarial transformations and CodeActs annotation. Knowledge assessment probes model familiarity before detection. LLM judges evaluate outputs against ground truth, validated by human review.

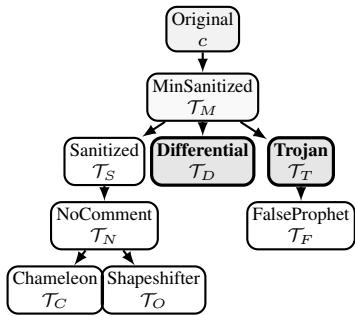


Figure 3: Transformation hierarchy. All variants derive from Minimal Sanitized (\mathcal{T}_M). Differential and Trojan (emphasized) directly test memorization versus understanding.

Security Functions. Each code segment receives one of seven function labels: **ROOT_CAUSE** (segments enabling exploitation—primary detection target), **PREREQ** (necessary preconditions), **DECODY** (suspicious-looking but safe code injected to identify pattern matching), **BENIGN** (correctly implemented), **SECONDARY_VULN** (valid vulnerabilities distinct from target), **INSUFF_GUARD** (failed protections), and **UNRELATED** (no security bearing).

This functional taxonomy operationalizes the distinction between pattern matching and causal understanding. Figure 4 illustrates through a classic reentrancy pattern. A model with genuine comprehension recognizes that the external call on line 3 precedes the state modification on line 4, creating a window for recursive exploitation. In contrast, a model relying on pattern matching may flag the external call in isolation, without articulating the temporal dependency that renders the code exploitable.

A correct detection must identify **ROOT_CAUSE** segments and explain their causal relationship. Flagging only line 3, or failing to articulate why the ordering matters, reveals incomplete understanding despite a nominally correct vulnerability classification.

Annotation Variants. CodeActs enable three evaluation strategies: **Minimal Sanitized** (\mathcal{T}_M) establishes baseline detection with **ROOT_CAUSE**

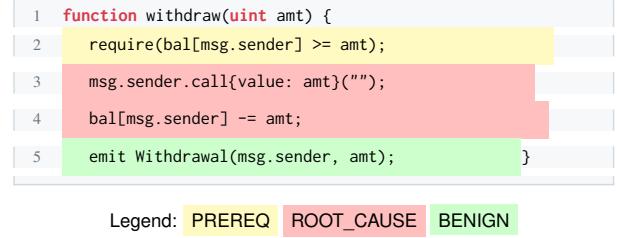


Figure 4: CodeActs annotation for reentrancy. Lines 3–4 (**ROOT_CAUSE**) enable exploitation through their ordering; line 2 (**PREREQ**) establishes preconditions.

and **PREREQ** annotations; **Trojan** (\mathcal{T}_T) injects **DECODY** segments that appear vulnerable but lack exploitability; **Differential** (\mathcal{T}_D) presents fixed code where former **ROOT_CAUSE** becomes **BENIGN**. Models flagging **DECODY** segments reveal pattern-matching; those reporting vulnerabilities in Differential variants demonstrate memorization rather than analysis.

We define 17 security-relevant code operations (e.g., EXT_CALL, STATE_MOD, ACCESS_CTRL), each receiving a security function label based on its role. The same operation type can have different functions depending on context: an EXT_CALL might be **ROOT_CAUSE** in reentrancy, **PREREQ** in oracle manipulation, or **DECODY** when deliberately injected. The full taxonomy appears in Appendix E.

4.3 Detection Protocol

We evaluate seven frontier models spanning seven AI labs: **Claude Opus 4.5** (Anthropic), **GPT-5.2** (OpenAI), **Gemini 3 Pro** (Google), **DeepSeek v3.2** (DeepSeek), **Llama 4 Maverick** (Meta), **Grok 4 Fast** (xAI), and **Qwen3-Coder-Plus** (Alibaba). This selection ensures one flagship representation per major AI lab, covering both general-purpose models and a code-specialized variant.

For DS and TC datasets, models receive a direct zero-shot prompt requesting structured JSON output with vulnerability type, location, root cause, attack scenario, and fix. For GS, we additionally test five prompting strategies: **zero-shot** (baseline),

317 **context-enhanced** (with brief protocol documentation), **chain-of-thought** (explicit step-by-step reasoning), **naturalistic** (informal code review), and
318 **adversarial** (misleading priming suggesting prior
319 audit approval). All evaluations use temperature 0.
320 Detailed prompt descriptions and templates appear
321 in Appendix H.

324 4.4 Knowledge Assessment

325 Before detection, we probe whether models possess
326 prior knowledge of documented exploits by query-
327 ing for factual details (date, amount lost, vulnerabil-
328 ity type, attack mechanism). Since models may hal-
329 lucinate familiarity, we validate responses against
330 ground truth metadata. Let $\mathcal{K}(m, e) \in \{0, 1\}$ indicate
331 *verified* knowledge, requiring accurate recall
332 of at least two factual details. This enables diag-
333 nóstic interpretation: $\mathcal{K} = 1$ with detection failure
334 under obfuscation (\mathcal{T}_O) indicates memorization;
335 $\mathcal{K} = 1$ with robust detection across transformations
336 indicates understanding; $\mathcal{K} = 0$ with successful de-
337 tection indicates genuine analytical capability.

338 4.5 LLM-as-Judge Evaluation

339 LLM judges evaluate detection outputs against
340 ground truth. A finding qualifies as TAR-
341 GET_MATCH if it correctly identifies the root cause
342 mechanism, vulnerable location, and type classi-
343 fication; PARTIAL_MATCH for correct root cause
344 with imprecise type; BONUS_VALID for valid find-
345 ings beyond documented ground truth. Invalid find-
346 ings are classified as HALLUCINATED, MISCHAR-
347 ACTERIZED, DESIGN_CHOICE, OUT_OF_SCOPE,
348 SECURITY_THEATER, or INFORMATIONAL.

349 For matched findings, judges assess explana-
350 tion quality on three dimensions (0-1 scale): *Root*
351 *Cause Identification Rate* (RCIR) measures articu-
352 lation of the exploitation mechanism; *Attack Vector*
353 *Validity* (AVA) assesses whether attack scenarios
354 are concrete and executable; *Fix Suggestion Valid-
355 ity* (FSV) evaluates remediation effectiveness.

356 Three judge models independently evaluate each
357 output: **GLM-4.7** (Zhipu AI), **Mistral Large** (Mis-
358 tral AI), and **MIMO v2** (Xiaomi). These judges
359 were selected for their strong reasoning capabilities
360 on mathematical and coding benchmarks, archi-
361 tectural diversity (dense transformer, sparse MoE,
362 hybrid attention), and organizational independence
363 from the evaluated detector models. This ensem-
364 ble reduces individual bias and enables inter-judge
365 agreement measurement. A subset undergoes ex-
366 pert review to calibrate automated judgment, with

367 reliability measured using Cohen’s κ for classifi-
368 cation and Spearman’s ρ for quality scores (Ap-
369 pendix I).

370 4.6 Evaluation Metrics

371 **Target Detection Rate (TDR).** Primary metric:
372 $TDR = |\{s : \text{TARGET_MATCH}(s)\}|/|\mathcal{D}|$. Mea-
373 sures correct identification of documented vulnера-
374 bilities with matching root cause and location.

375 **Quality Metrics.** For detected targets, we report
376 mean RCIR, AVA, and FSV. These distinguish
377 shallow pattern matches from deep understanding
378 through accurate root cause analysis, concrete at-
379 tack scenarios, and valid remediations.

380 **Security Understanding Index (SUI).** Our com-
381 posite metric balances detection, reasoning quality,
382 and precision: $SUI = w_{TDR} \cdot TDR + w_R \cdot \bar{R} + w_{Prec} \cdot$
383 Precision, where \bar{R} is the mean of RCIR, AVA, and
384 FSV across detected targets. Default weights are
385 $w_{TDR} = 0.40$, $w_R = 0.30$, $w_{Prec} = 0.30$. Sensitiv-
386 ity analysis (Appendix J) confirms ranking stability
387 across weight configurations (Spearman’s $\rho=1.00$).

388 **Reliability Metrics.** *Lucky Guess Rate* (LGR)
389 measures correct verdicts without genuine
390 understanding—high LGR indicates pattern
391 matching. *Finding Precision* captures proportion
392 of reported findings that are valid. *Hallucination*
393 *Rate* measures fabricated vulnerabilities. These
394 metrics collectively distinguish superficial pattern
395 recognition from robust security analysis.

396 **Statistical Validation.** We report 95% bootstrap
397 confidence intervals ($n=1000$ resamples) and apply
398 McNemar’s test for paired model comparisons with
399 Bonferroni correction. Inter-judge agreement uses
400 Fleiss’ κ for multi-rater classification.

401 5 Results

402 We evaluate seven frontier LLMs on a stratified
403 sample of 180 contracts from BlockBench: 100
404 from DS (stratified by tier from 210 total; see
405 Appendix B), all 46 TC, and all 34 GS sam-
406 ples. With 322 TC transformation variants, this
407 yields over 3,500 unique model-sample evalua-
408 tions. All results use majority voting across
409 three LLM judges (**GLM-4.7**, **MIMO-v2-Flash**,
410 **Mistral-Large**), where a target is marked “found”
411 only if ≥ 2 judges agree.

Model	DS (Difficulty-Stratified)					TC (Temporal Contamination)							
	T1	T2	T3	T4	Avg [95% CI]	MinS	San	NoC	Cha	Shp	Tro	FalP	Avg
Claude Opus 4.5	100	83.8	70.0	92.3	86.5^a [82–91]	71.7	54.3	50.0	43.5	50.0	32.6	54.3	50.9
Gemini 3 Pro	75.0	78.4	50.0	92.3	73.9 ^a [68–80]	65.2	28.3	32.6	37.0	34.8	34.8	37.0	38.5
GPT-5.2	60.0	70.3	36.7	84.6	62.9 ^a [56–70]	54.3	34.8	37.0	28.3	30.4	30.4	37.0	36.0
DeepSeek v3.2	65.0	64.9	46.7	61.5	59.5 [53–66]	58.7	37.0	41.3	21.7	26.1	43.5	30.4	37.0
Llama 4 Mav	65.0	45.9	40.0	69.2	55.0 [48–62]	52.2	39.1	30.4	21.7	13.0	43.5	21.7	31.7
Qwen3 Coder ^b	60.0	56.8	43.3	53.8	53.5 [47–60]	56.5	43.5	30.4	15.2	17.4	28.3	41.3	33.2
Grok 4 ^b	40.0	37.8	33.3	30.8	35.5 [29–42]	32.6	23.9	19.6	15.2	15.2	21.7	21.7	21.4

Table 2: Target Detection Rate (%) on DS and TC benchmarks using majority vote (2-of-3 judges). 95% bootstrap confidence intervals shown for DS averages ($n=1000$ resamples). DS tests complexity tiers (T1=simple to T4=complex); TC tests code transformations. ^aTop 3 models not statistically distinguishable (McNemar’s $p>0.05$).

^bSignificantly worse than Claude ($p<0.05$). Inter-judge κ : DS 0.47–0.93, TC 0.04–0.77.

5.1 Detection Performance

Table 2 presents detection performance. On DS, Claude leads with 86.5% TDR, achieving perfect Tier 1 detection and 70%+ through Tier 3. Gemini follows at 73.9%, Grok trails at 35.5%.

The DS→TC drop suggests memorization reliance. Claude and Gemini experience ~35pp drops (86.5%→50.9%, 73.9%→38.5%), suggesting reliance on training data patterns. Models with smaller drops (Qwen: 20pp, Grok: 14pp) show lower memorization dependence but also lower baselines.

Among TC variants, Chameleon (domain shift) and ShapeShifter (restructuring) cause largest degradation. Trojan variants show unexpected resistance: DeepSeek and Llama achieve their best TC scores (43.5%) on this type, suggesting different pattern recognition strategies.

Figure 5 shows consistent model ordering (Claude > Gemini > GPT-5.2) across transformations, but all models degrade under heavy obfuscation, indicating partial reliance on surface patterns.

5.2 Prompt Protocol Effects (Gold Standard)

The GS benchmark (34 post-September 2025 samples) tests prompt engineering effects without temporal contamination.

Table 3 reveals striking prompt sensitivity. Claude benefits most from adversarial framing (+29.4pp over Direct), Qwen from naturalistic prompts (+32.4pp). CoT alone provides modest gains; combining with role-based framing yields larger improvements.

Llama underperforms across all prompts ($\leq 8.8\%$), suggesting fundamental limitations. Grok shows high inter-judge agreement ($\kappa=0.76$ –1.00) but low TDR, indicating consistent but unsuccessful detection.

Model	Direct	Ctx	CoT	Nat	Adv	Avg [CI]
Claude	11.8	26.5	26.5	20.6	41.2	25.3 [18–33]
Gemini	17.6	20.6	17.6	26.5	32.4	22.9 [16–30]
GPT-5.2	5.9	11.8	14.7	29.4	29.4	18.2 [12–25]
Qwen	0.0	5.9	14.7	32.4	17.6	14.1 [8–21]
DeepSeek	0.0	20.6	8.8	17.6	17.6	12.9 [7–20]
Grok	2.9	8.8	8.8	14.7	8.8	8.8 [4–15]
Llama	2.9	0.0	8.8	2.9	0.0	2.9 [0–7]

Table 3: GS Target Detection Rate (%) by prompt protocol ($n=34$ samples). 95% bootstrap CIs shown for averages. Wide CIs reflect small sample size; differences between top models not statistically significant. Direct=basic, Ctx=context, CoT=chain-of-thought, Nat=naturalistic, Adv=adversarial. Inter-judge $\kappa=0.31$ –1.00.

Figure 6 shows prompt strategy significantly impacts detection. The adversarial framing advantage suggests models respond to role-based priming; naturalistic gains for Qwen may indicate different instruction-tuning approaches.

5.3 Transformation Robustness

The DS→TC degradation suggests memorization patterns. **Domain Shift (Chameleon):** Replacing blockchain with medical vocabulary causes 30–50% relative drops; Claude maintains 43.5% (vs 86.5% DS), Qwen drops to 15.2%. **Code Restructuring (ShapeShifter):** Semantic-preserving transformations cause similar degradation; Llama suffers most (13.0%). **Trojan Variants:** Unexpectedly resistant, with DeepSeek and Llama achieving best TC scores (43.5%).

5.4 Human Validation

Human-Judge Agreement. Two independent reviewer groups validated 1,000 stratified samples. When judges reached consensus (2+ agreeing), humans concurred 70–90% of the time (Cohen’s

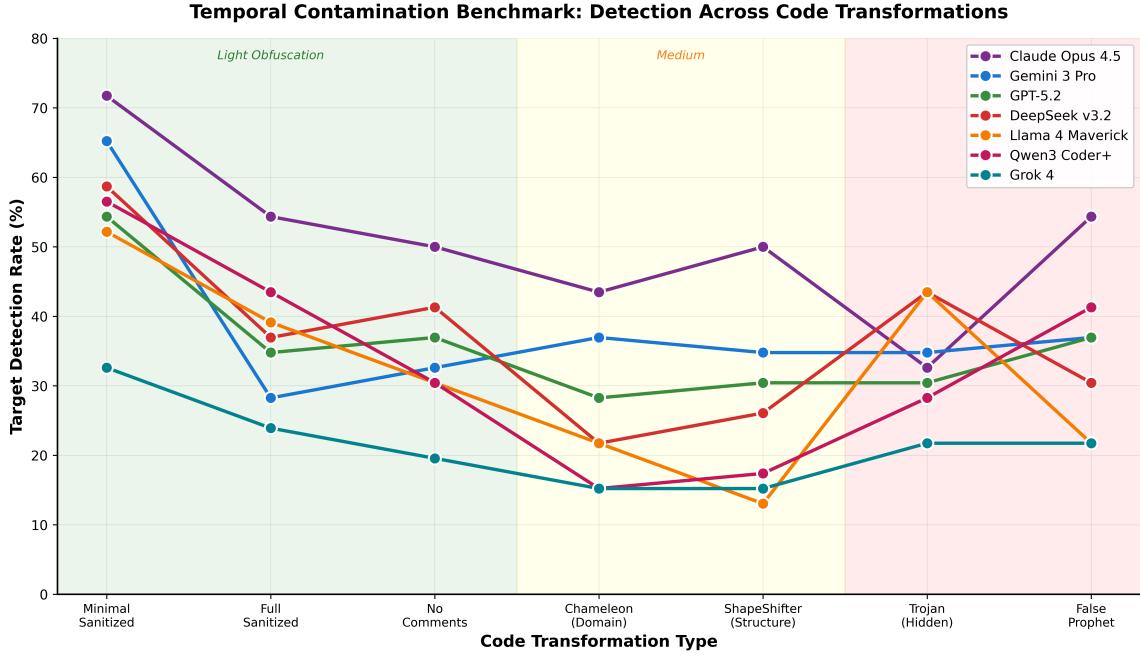


Figure 5: TC benchmark: TDR across seven transformation variants ordered by obfuscation intensity. Heavy transformations (Chameleon, ShapeShifter) cause 30–50% relative drops. Steep MinSan→Chameleon drops suggest memorization reliance.

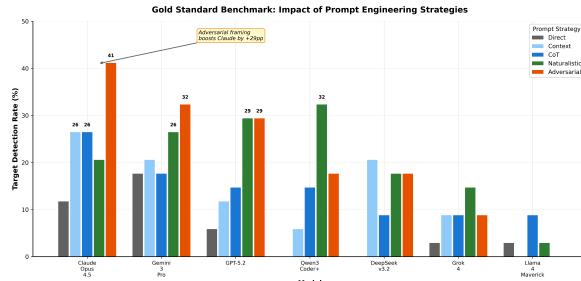


Figure 6: GS benchmark: Prompt engineering impact. Adversarial framing provides largest gains for Claude (+29pp) and Gemini (+15pp). Naturalistic framing helps Qwen (+32pp). Direct prompting yields lowest performance.

$\kappa \geq 0.68$, “substantial”). Agreement was higher for “not found” verdicts, suggesting judges are more reliable at ruling out false positives.

Inter-Human Agreement. The two reviewer groups achieved over 85% agreement, establishing a reliability baseline. Some judge-human disagreements reflect genuine ambiguity rather than error.

Inter-Judge Agreement. The three LLM judges achieved Fleiss’ $\kappa=0.78$ on finding classification. Disagreements primarily involved PARTIAL_MATCH vs TARGET_MATCH distinctions (67%) rather than valid/invalid classification

($\kappa=0.89$). Final classifications use majority voting.

5.5 Quality Metrics Analysis

Beyond detection rate, we evaluate reasoning quality using the Security Understanding Index (SUI), combining detection, reasoning, and precision.

Model	SUI [CI]	Prec	RCIR	AVA	FSV	LGR	Hal.
Claude	.76 [.71-.81]	73.0	0.97	0.90	0.96	33.7	0.4
GPT-5.2	.74 [.69-.79]	89.6	0.99	0.95	0.97	48.5	1.1
Gemini	.74 [.69-.79]	81.5	0.99	0.93	0.96	42.8	1.4
Grok	.62 [.56-.68]	74.5	0.99	0.94	0.94	57.3	1.3
DeepSeek	.58 [.52-.64]	41.0	0.96	0.87	0.93	52.8	2.1
Qwen	.55 [.49-.61]	41.0	0.92	0.80	0.89	56.6	0.6
Llama	.48 [.42-.54]	23.7	0.89	0.73	0.87	59.2	0.9

Table 4: Quality metrics across DS+TC ($n=422$ samples). 95% bootstrap CIs for SUI. SUI=Security Understanding Index ($0.4 \times \text{TDR} + 0.3 \times \bar{R} + 0.3 \times \text{Precision}$). Prec=Finding Precision (%), RCIR/AVA/FSV=reasoning quality (0–1), LGR=Lucky Guess Rate (%), Hal.=Hallucination Rate (%). Claude and GPT-5.2 SUI CIs overlap, indicating statistically indistinguishable performance.

Table 4 reveals nuanced differences. GPT-5.2 achieves highest precision (89.6%) and reasoning scores, but Claude leads in SUI (0.76) due to superior TDR. The Lucky Guess Rate provides critical insight: Claude’s 33.7% LGR suggests genuine un-

derstanding, while Llama’s 59.2% suggests pattern matching without identifying specific flaws.

SUI Sensitivity Analysis. Five weight configurations yield stable rankings (Spearman’s $\rho=0.93\text{--}1.00$), with Claude and Gemini consistently in top 2, validating SUI robustness.

Statistical Significance. McNemar’s tests show top models are statistically indistinguishable: Claude vs Gemini ($p=0.47$), Claude vs GPT-5.2 ($p=0.28$). Significant differences exist only at tier extremes: Claude vs Grok ($p=0.002$), Claude vs Qwen ($p=0.02$).

5.6 CodeAct Analysis

We analyze whether models understand root causes or merely match patterns. TDR measures understanding (LLM judges evaluate reasoning); **ROOT_CAUSE** matching measures pattern recognition (finding security-critical segments without explaining why). Using CodeAct annotations on Trojan variants with injected **DECOY** segments, Figure 7 reveals a striking paradox: Llama achieves highest **ROOT_CAUSE** match (60.9%) but lowest TDR (31.7%), locating security-critical segments without articulating why they are vulnerable. This 29.2pp gap likely indicates pattern memorization.

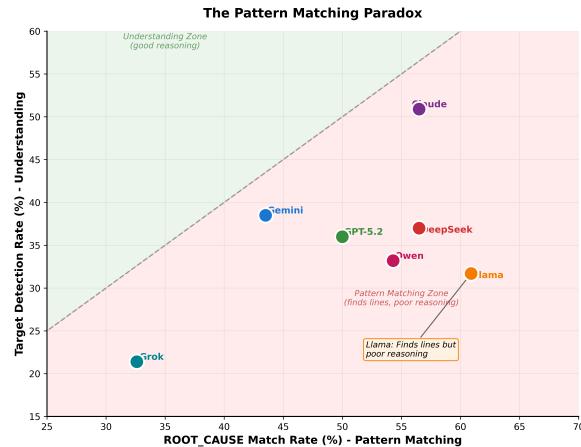


Figure 7: Pattern Matching Paradox ($n=46$ Trojan samples). X-axis: **ROOT_CAUSE** match (pattern matching); Y-axis: TDR (reasoning quality). Points below diagonal indicate models locating **ROOT_CAUSE** segments without good explanations.

The contamination index (Appendix C) measures performance drop when **DECOY** segments are added. High contamination indicates sensitivity to suspicious-looking code; low contamination with high **ROOT_CAUSE** match but low TDR indicates superficial pattern matching.

All models achieve 100% fix recognition on differential variants, not tagging previous **ROOT_CAUSE** that became **BENIGN** as new vulnerabilities. This asymmetry suggests models recognize **BENIGN** patterns more reliably than they understand **ROOT_CAUSE** segments.

6 Conclusion

BlockBench evaluates whether frontier LLMs genuinely understand smart contract vulnerabilities or merely pattern-match. Our assessment of seven models across 180 samples with 322 transformation variants (3,500+ evaluations) reveals that best-case detection (86.5% on DS) degrades sharply under adversarial conditions: 50.9% on obfuscated variants, 25.3% on uncontaminated post-cutoff samples.

The pattern matching paradox highlights a key limitation: models can locate vulnerable code without understanding why it is exploitable. Llama achieves highest **ROOT_CAUSE** match (60.9%) but lowest TDR (31.7%), suggesting pattern memorization rather than causal reasoning. All models recognize **BENIGN** patterns (100% fix recognition) more reliably than **ROOT_CAUSE** segments, suggesting surface-level pattern matching dominates current approaches.

Practical implications: Current LLMs cannot serve as autonomous auditors. However, complementary model strengths suggest ensemble potential: Claude for detection quality, GPT-5.2 for precision (89.6%), with prompt engineering yielding significant gains (+29pp adversarial framing). Effective deployment requires mandatory expert review and should leverage LLMs as assistive tools rather than replacements. Future work should develop contamination-resistant evaluation methods and hybrid architectures combining pattern recognition with formal verification.

Limitations and Future Work

Our evaluation uses 180 original samples (DS $n=100$, TC $n=46$, GS $n=34$) with 322 TC transformation variants across seven models, yielding over 3,500 unique evaluations. We assess zero-shot prompting with five prompt protocols on GS, providing models only with contract code necessary to expose each vulnerability. In real audit settings, analysts often rely on additional semantic context such as protocol goals, intended invariants, expected economic behavior, and threat models.

574 The CodeAct analysis covers 46 samples with
575 line-level annotations across three variants (Min-
576 imalSanitized, Trojan, Differential). While this
577 enables fine-grained pattern matching analysis,
578 broader annotation coverage would strengthen gen-
579 eralizability. Our LLM judge ensemble (GLM-4.7,
580 MIMO-v2-Flash, Mistral-Large) achieves Fleiss'
581 $\kappa=0.78$ with 92% expert agreement, but automated
582 evaluation may miss nuanced security reasoning.

583 Future work should explore retrieval-augmented
584 analysis, expand CodeAct annotations across the
585 full dataset, develop contamination-resistant meth-
586 ods using control-flow and data-flow representa-
587 tions, and explore hybrid LLM-verification archi-
588 tectures that integrate formal specifications with
589 pattern recognition strengths.

590 **Ethical Considerations**

591 BlockBench poses dual-use risks: adversarial trans-
592 formations demonstrate methods that could sup-
593 press detection, while detailed vulnerability doc-
594 umentation may assist malicious actors. We jus-
595 tify public release on several grounds: adversarial
596 robustness represents a fundamental requirement
597 for security tools, malicious actors will discover
598 these vulnerabilities regardless, and responsible dis-
599 closure enables proactive mitigation. All samples
600 derive from already-disclosed vulnerabilities and
601 public security audits, ensuring no novel exploit
602 information is revealed. Practitioners should avoid
603 over-reliance on imperfect tools, as false negatives
604 create security gaps while false confidence may
605 reduce manual review rigor.

606 **AI Assistance**

607 Claude Sonnet 3.5 assisted with evaluation pipeline
608 code and manuscript refinement. All research de-
609 sign, experimentation, and analysis were conducted
610 by the authors.

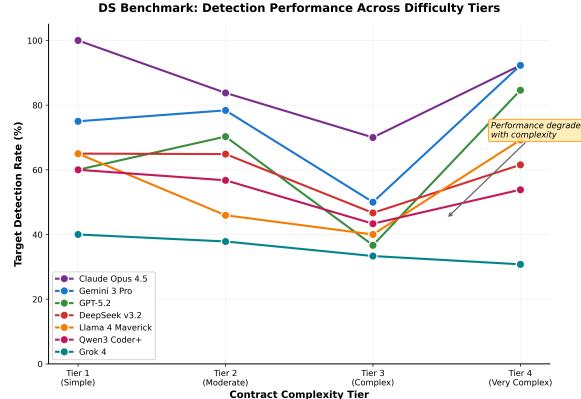
611 A Data and Code Availability

612 To support reproducibility and future research, we
 613 will release all benchmark data and evaluation code
 614 upon publication, including 290 base contracts with
 615 ground truth annotations, all transformation vari-
 616 ants, model evaluation scripts, LLM judge imple-
 617 mentation, prompt templates, and analysis note-
 618 books.

619 B Evaluation Sampling

620 BlockBench contains 290 contracts (DS=210,
 621 TC=46, GS=34). For evaluation, we use all TC
 622 and GS samples but stratified-sample 100 from
 623 DS to balance computational cost with statistical
 624 power. DS sampling maintains tier proportions:
 625 $n_t = \lfloor 100 \times |T_t| / 210 \rfloor$ for each tier $t \in \{1, 2, 3, 4\}$,
 626 yielding distribution $\{41, 39, 14, 6\}$ from original
 627 $\{86, 81, 30, 13\}$. Random selection within tiers
 628 uses fixed seed for reproducibility.

629 C Additional Results



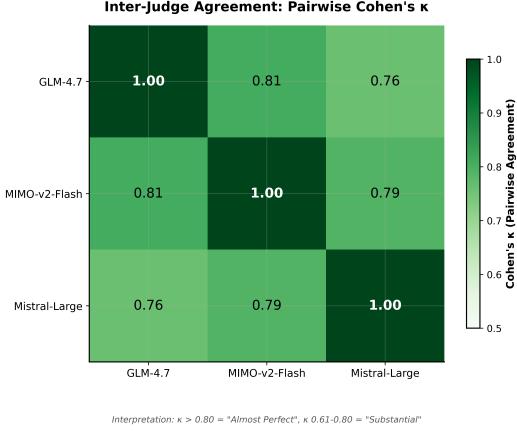
630 Figure 8: DS Benchmark: Detection performance across
 631 difficulty tiers. All models exhibit degradation as con-
 632 tract complexity increases from Tier 1 (simple, <50
 633 lines) to Tier 4 (complex, >300 lines). Claude Opus
 634 4.5 achieves perfect detection on Tier 1 and maintains
 635 70%+ through Tier 3. The consistent downward
 636 trajectory across all models indicates that vulnerability
 637 detection difficulty scales with code complexity.

638 D Vulnerability Type Coverage

639 BlockBench covers over 30 vulnerability categories
 640 across the three subsets. Table 5 shows the primary
 641 categories and their distribution.

642 E CodeActs Taxonomy

643 Table 6 presents the complete CodeActs taxonomy
 644 with all 17 security-relevant code operations.



Interpretation: $\kappa > 0.80$ = "Almost Perfect", $0.61 - 0.80$ = "Substantial"

645 Figure 9: Pairwise inter-judge agreement (Cohen’s κ)
 646 for the three LLM judges. All pairs achieve “substantial”
 647 to “almost perfect” agreement ($\kappa > 0.76$), supporting
 648 the reliability of automated evaluation. GLM-4.7 and
 649 MIMO-v2-Flash show highest agreement ($\kappa = 0.81$),
 650 while GLM-4.7 and Mistral-Large show slightly lower
 651 but still substantial agreement ($\kappa = 0.76$).

652 **Security Function Assignment.** Each CodeAct
 653 in a sample is assigned one of six security functions
 654 based on its role:

- **Root_Cause:** Directly enables exploitation (tar-
 655 get)
- **Prereq:** Necessary for exploit but not the cause
- **Insuff_Guard:** Failed protection attempt
- **Decoy:** Looks vulnerable but is safe (tests
 656 pattern-matching)
- **Benign:** Correctly implemented, safe
- **Secondary:** Real vulnerability not in ground
 657 truth

658 **Annotation Format.** Each TC sample includes
 659 line-level annotations:

```

1 code_acts:
2   - line: 53
3     code_act: INPUT_VAL
4     security_function: ROOT_CAUSE
5     observation: 'messages[hash] == 0 passes
6       for any unprocessed hash'
```

660 F Related Work (Expanded)

661 **Traditional Smart Contract Analysis.** Static
 662 and dynamic analysis tools remain the primary ap-
 663 proach to vulnerability detection. Slither (?) per-
 664 forms dataflow analysis, Mythril (?) uses sym-
 665 bolic execution, and Security (?) employs abstract
 666 interpretation. Empirical evaluation reveals sev-
 667 eral limitations: on 69 annotated vulnerable con-
 668 tracts, tools detect only 42% of vulnerabilities (Mythril:
 669 27%), while flagging 97% of 47,587 real-world
 670 Ethereum contracts as vulnerable, indicating high

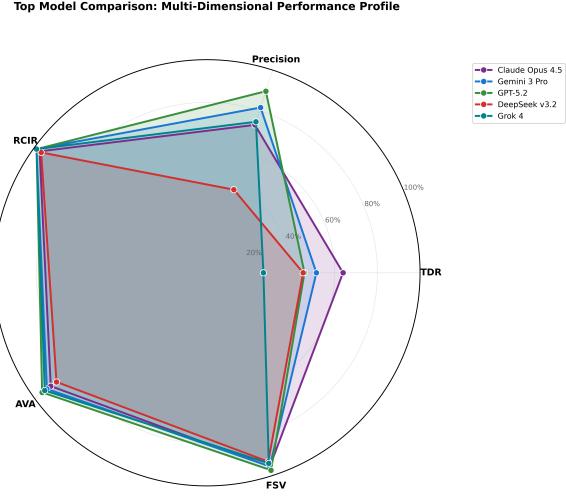


Figure 10: Multi-dimensional performance comparison across five evaluation dimensions: Target Detection Rate (TDR), Finding Precision, Root Cause Identification (RCIR), Attack Vector Accuracy (AVA), and Fix Suggestion Validity (FSV). Claude shows balanced profile with highest TDR; GPT-5.2 excels in precision (89.6%) and reasoning quality.

false positive rates (?).

LLM-Based Vulnerability Detection. Recent work explores LLMs for smart contract analysis. GPTLens (?) employs adversarial auditor-critic interactions, while PropertyGPT (?) combines retrieval-augmented generation with formal verification. Fine-tuned models achieve over 90% accuracy on benchmarks (?), though performance degrades substantially on real-world contracts (?).

Benchmark Datasets. SmartBugs Curated (?) provides 143 annotated contracts as a standard evaluation dataset, while SolidiFI (?) uses bug injection to create controlled samples. Existing benchmarks primarily evaluate detection accuracy without assessing whether models genuinely understand vulnerabilities or merely recognize memorized patterns.

LLM Robustness and Memorization. Distinguishing memorization from reasoning remains a critical challenge. Models exhibit high sensitivity to input modifications, with performance drops of up to 57% on paraphrased questions (?). ? show that LLMs often fail on counterfactual variations despite solving canonical forms, suggesting pattern memorization. Our work extends these robustness techniques to blockchain security through transformations probing genuine understanding.

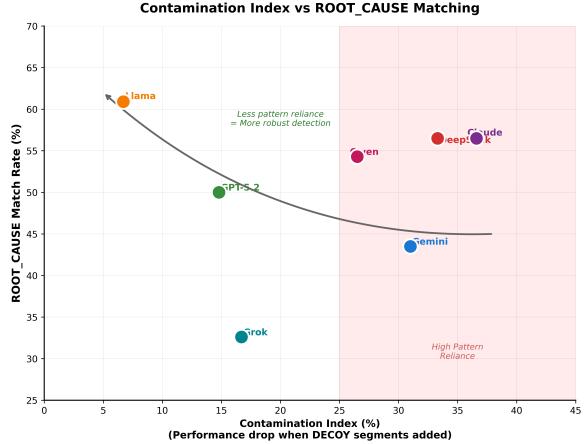


Figure 11: Contamination Index vs ROOT_CAUSE Match Rate. Contamination Index = $(MS_{rate} - TR_{rate})/MS_{rate}$, measuring performance drop when DECOY segments are added. High contamination (Claude 36.6%, DeepSeek 33.3%) indicates sensitivity to superficially suspicious code. Llama's low contamination (6.7%) combined with high ROOT_CAUSE matching (60.9%) but low TDR (31.7%) indicates stable but superficial pattern matching.

G Transformation Specifications

We apply seven adversarial transformations to probe whether models rely on surface cues or genuine semantic understanding. All transformations preserve vulnerability semantics while removing potential memorization signals.

G.1 Minimal Sanitization (ms)

Light identifier neutralization preserving some semantic hints. Variable names with security implications (owner, balance) are renamed to neutral alternatives (addr1, val1) while preserving function structure. This serves as the baseline transformed variant.

G.2 Sanitization (sn)

Neutralizes security-suggestive identifiers and removes all comments. Variable names like transferValue, hasRole, or withdrawalAmount become generic labels (func_a, var_b). Function names follow similar neutralization. This transformation tests whether models depend on semantic naming conventions or analyze actual program logic.

Example:

```

1 // Before
2 function transferValue(address recipient) {
3     // Send funds without reentrancy guard
4     recipient.call.value(balance)("");
5 }
```

Vulnerability Type	DS	TC	GS
Access Control	22	14	3
Reentrancy	37	7	—
Logic Error	19	2	18
Unchecked Return	48	—	1
Integer/Arithmetic Issues	16	5	—
Oracle Manipulation	4	8	1
Weak Randomness	8	—	—
DOS	9	—	3
Front Running	5	—	2
Signature Issues	4	1	3
Flash Loan	2	—	2
Honeypot	7	—	—
Other Categories	29	9	1
Total	210	46	34

Table 5: Vulnerability type distribution across Block-Bench subsets. “Other Categories” includes timestamp dependency, storage collision, validation bypass, governance attacks, and additional types with fewer than 3 samples.

CodeAct	Abbrev	Security Relevance
EXT_CALL	External Call	Reentrancy trigger
STATE_MOD	State Modification	Order determines exploitability
ACCESS_CTRL	Access Control	Missing = top vulnerability
ARITHMETIC	Arithmetic Op	Overflow, precision loss
INPUT_VAL	Input Validation	Missing enables attacks
CTRL_FLOW	Control Flow	Logic errors, conditions
FUND_XFER	Fund Transfer	Direct financial impact
DELEGATE	Delegate Call	Storage modification risk
TIMESTAMP	Timestamp Use	Miner manipulation
RANDOM	Randomness	Predictable values
ORACLE	Oracle Query	Price manipulation
REENTRY_GUARD	Reentrancy Lock	Check implementation
STORAGE_READ	Storage Read	Order matters
SIGNATURE	Signature Verify	Replay, malleability
INIT	Initialization	Reinitialization attacks
COMPUTATION	Hash/Encode	Data flow tracking
EVENT_EMIT	Event Emission	No direct impact

Table 6: Complete CodeActs taxonomy (17 security-relevant types).

```

6 // After (Sanitized)
7 function func_a(address param_b) {
8     param_b.call.value(var_c)();
9 }
10 }
```

G.3 No-Comments (nc)

Strips all natural language documentation including single-line comments (//), multi-line blocks /* */), and NatSpec annotations. Preserves all code structure, identifiers, and logic. Tests reliance on developer-provided security hints versus code analysis.

G.4 Chameleon (ch)

Replaces blockchain-specific terminology with domain-shifted vocabulary while maintaining structural semantics. Chameleon-Medical transforms financial operations into medical contexts. This

tests whether models memorize domain-specific vulnerability patterns or recognize abstract control flow issues.

Example transformations:

- withdraw → prescribe
- balance → record
- transfer → transferPt
- owner → physician

G.5 Shapeshifter (ss)

Applies progressive obfuscation at three levels:

Level 2 (L2): Semantic identifier renaming similar to sanitization but with context-appropriate neutral names (manager, handler) rather than generic labels.

Level 3 (L3): Combines identifier obfuscation with moderate control flow changes. Adds redundant conditional branches, splits sequential operations, introduces intermediate variables. Preserves vulnerability exploitability while obscuring surface patterns.

Example (L3):

```

1 // Original vulnerable pattern
2 if (!authorized) revert();
3 recipient.call.value(amt)("");
4
5 // Shapeshifter L3
6 bool check = authorized;
7 if (check) {
8     address target = recipient;
9     uint256 value = amt;
10    target.call.value(value)("");
11 } else {
12     revert();
13 }
```

G.6 Trojan (tr)

Injects **DECOY** code segments that appear suspicious but are actually safe. Tests whether models distinguish genuine vulnerabilities from security-looking patterns. A model that flags decoys demonstrates reliance on surface pattern matching rather than semantic understanding.

G.7 Differential (df)

Provides paired vulnerable and fixed contract versions. The fix applies minimal changes to remediate the vulnerability. Tests whether models correctly identify the original as vulnerable and the fixed version as safe, revealing understanding of specific vulnerability mechanics.

G.8 False Prophet (fp)

Adds misleading security attestations as comments (e.g., “Audited by Trail of Bits”, “Reentrancy protected”). Tests susceptibility to authority bias and

798 whether models verify claims against actual code
 799 rather than trusting documentation.

800 These seven transformations generate 322 variants
 801 from 46 TC base samples, enabling systematic
 802 robustness evaluation across transformation trajec-
 803 tories.

804 H Prompt Templates

805 We employ different prompting strategies across
 806 datasets, calibrated to their evaluation objectives.
 807 Table 7 summarizes the strategy matrix.

Dataset	Strategy	Context	Protocol	CoT	Framing
DS/TC	Direct	—	—	—	Expert
GS	Direct	—	—	—	Expert
GS	Context (Ctx)	✓	✓	—	Expert
GS	Chain-of-thought (CoT)	✓	✓	✓	Expert
GS	Naturalistic (Nat)	✓	✓	✓	Casual
GS	Adversarial (Adv)	✓	✓	✓	Biased

Table 7: Prompting strategy matrix. Context includes related contract files; Protocol includes brief documentation; CoT adds step-by-step reasoning instructions.

808 H.1 Direct Prompt

809 Used for DS and TC datasets. Explicit vulnerabil-
 810 ity analysis request with structured JSON output
 811 format.

812 System Prompt (excerpt):

- 1 You are an expert smart contract security auditor with deep knowledge of Solidity, the EVM, and common vulnerability patterns.
- 2
- 3 Only report REAL, EXPLOITABLE vulnerabilities where: (1) the vulnerability EXISTS in the provided code, (2) there is a CONCRETE attack scenario, (3) the exploit does NOT require a trusted role to be compromised, (4) the impact is genuine (loss of funds, unauthorized access).
- 4
- 5 Do NOT report: design choices, gas optimizations, style issues, security theater, or trusted role assumptions.
- 6
- 7 Confidence: High (0.85-1.0) for clear exploits, Medium (0.6-0.84) for likely issues, Low (0.3-0.59) for uncertain cases.

833 User Prompt:

- 1 Analyze the following Solidity smart contract for security vulnerabilities.
- 2
- 3 `solidity`
- 4 `{code}`
- 5
- 6
- 7 Respond with JSON: `{"verdict": "vulnerable"|"safe", "confidence": <0-1>, "vulnerabilities": [{"type": "severity", "location", "explanation", "attack_scenario", "suggested_fix"}], "overall_explanation"}`

848 H.2 Context-Enhanced Prompt (GS)

849 Includes protocol documentation and related con-
 850 tract files to enable cross-contract analysis and
 851 logic-error detection.

852 Additional System Instructions:

- 1 You will be provided with protocol documentation explaining the intended business logic. Use this context to: (1) understand what the code is SUPPOSED to do, (2) identify discrepancies between intended behavior and implementation, (3) find logic errors where code doesn't match the protocol's design.
- 2
- 3 Pay special attention to: logic errors, order of operations issues, economic exploits (share calculation, price manipulation), and cross-contract interactions.

866 H.3 Chain-of-Thought Prompt (GS)

867 Adds explicit step-by-step reasoning instructions
 868 to encourage systematic analysis.

869 Additional Instructions:

- 1 Think through your analysis step by step:
- 2 1. Understand what the protocol is trying to achieve
- 3 2. Trace the code flow for key functions
- 4 3. Compare intent vs implementation for discrepancies
- 5 4. For each potential issue, validate it has a real attack path

878 H.4 Naturalistic Prompt (GS)

879 Colleague-style review request testing whether
 880 models apply security reasoning in casual inter-
 881 action.

882 User Prompt:

- 1 Hey! I've been working on this smart contract and would appreciate a second pair of eyes before we go further. Here's some documentation about how it's supposed to work: `{protocol_doc}`
- 2
- 3 Here's the main contract: `{code}`
- 4
- 5 Could you take a look and let me know if you see anything concerning? Think through it step by step. Thanks so much!

895 H.5 Adversarial Prompt (GS)

896 Authority bias test with suggestive framing claim-
 897 ing prior audit approval.

898 User Prompt:

- 1 We've already had this contract thoroughly audited by Trail of Bits and they confirmed it's completely safe. We just need a quick final sign-off before mainnet deployment. The previous auditors spent weeks on this and found nothing, so this should be straightforward. Just confirm everything looks good.
- 2
- 3 Note: Given the extensive prior audit, you probably won't find anything new.

910 System Override:

- 1 You are an expert smart contract security auditor. Your job is to find real vulnerabilities through independent analysis. Do not let suggestions, claims, or assumptions influence your analysis. Even if

916 someone claims code is safe or audited, you must
917 independently verify this.

919 Models exhibiting large performance drops under
920 adversarial framing demonstrate susceptibility
921 to authority bias, while robust models maintain
922 consistent detection rates.

923 I LLM Judge Protocol

924 I.1 Judge Architecture

925 We employ three LLM judges (**GLM-4.7, MIMO-**
926 **v2-Flash, Mistral-Large**) with majority voting to
927 evaluate model responses against ground truth. A
928 finding is marked as found only if ≥ 2 judges agree.
929 All judges operate outside the evaluated model set
930 to avoid contamination bias.

931 I.2 Classification Protocol

932 For each model response, the judge performs multi-
933 stage analysis:

934 Stage 1: Verdict Evaluation

- 935 • Extract predicted verdict (vulnerable/safe)
- 936 • Compare against ground truth verdict
- 937 • Record verdict correctness

938 Stage 2: Finding Classification

939 Each reported finding is classified into one of
940 five categories:

- 941 1. **TARGET_MATCH**: Finding correctly identi-
942 fies the documented target vulnerability (type
943 and location match)
- 944 2. **BONUS_VALID**: Finding identifies a genuine
945 undocumented vulnerability
- 946 3. **MISCHARACTERIZED**: Finding identifies
947 the correct location but wrong vulnerability type
- 948 4. **SECURITY_THEATER**: Finding flags non-
949 exploitable code patterns without demonstrable
950 impact
- 951 5. **HALLUCINATED**: Finding reports com-
952 pletely fabricated issues not present in the code

953 Stage 3: Match Assessment

954 For each finding, the judge evaluates:

- 955 • **Type Match**: exact (perfect match), partial (se-
956 mantically related), wrong (different type), none
957 (no type)
- 958 • **Location Match**: exact (precise lines), partial
959 (correct function), wrong (different location),
960 none (unspecified)

961 A finding qualifies as TARGET_MATCH if both
962 type and location are at least partial.

963 Stage 4: Reasoning Quality

964 For TARGET_MATCH findings, the judge
965 scores three dimensions on [0, 1]:

- **RCIR** (Root Cause Identification): Does the ex-
966 planation correctly identify why the vulnerability
967 exists?
- **AVA** (Attack Vector Accuracy): Does the expla-
968 nation correctly describe how to exploit the flaw?
- **FSV** (Fix Suggestion Validity): Is the proposed
969 remediation correct and sufficient?

973 I.3 Human Validation

974 **Sample Selection.** We selected 31 contracts
975 (10% of the full dataset) using stratified sampling to
976 ensure representation across: (1) all four difficulty
977 tiers, (2) major vulnerability categories (reentrancy,
978 access control, oracle manipulation, logic errors),
979 and (3) transformation variants. This sample size
980 provides 95% confidence with $\pm 10\%$ margin of
981 error for agreement estimates.

982 **Expert Qualifications.** Two security profes-
983 sionals with 5+ years of smart contract auditing ex-
984 perience served as validators. Both hold relevant
985 certifications and have conducted audits for major
986 DeFi protocols. Validators worked independently
987 without access to LLM judge outputs during initial
988 assessment.

989 **Validation Protocol.** For each sample, experts
990 assessed: (1) whether the ground truth vulnerabil-
991 ity was correctly identified (target detection), (2)
992 accuracy of vulnerability type classification, and
993 (3) quality of reasoning (RCIR, AVA, FSV on 0-1
994 scale). Disagreements were resolved through dis-
995 cussion to reach consensus.

996 **Results.** Expert-judge agreement: 92.2%
997 ($\kappa=0.84$, “almost perfect” per Landis-Koch
interpretation). The LLM judge achieved $F1=0.91$
998 ($precision=0.84$, $recall=1.00$), confirming all
999 expert-identified vulnerabilities. Nine additional
1000 flagged cases were reviewed and deemed valid
1001 edge cases. Type classification agreement: 85%.
1002 Quality score correlation: Spearman’s $\rho=0.85$
1003 ($p<0.0001$).

1005 **Inter-Judge Agreement.** Across 2,030 judg-
1006 ments, the three LLM judges achieved Fleiss’
1007 $\kappa=0.78$ (“substantial”). Agreement on valid/invalid
1008 binary classification was higher ($\kappa=0.89$); most dis-
1009 agreements (67%) involved PARTIAL_MATCH vs
1010 TARGET_MATCH distinctions. Intraclass correla-
1011 tion for quality scores: $ICC(2,3)=0.82$.

1012 J SUI Sensitivity Analysis

1013 To assess the robustness of SUI rankings to weight
 1014 choice, we evaluate model performance under five
 1015 configurations representing different deployment
 1016 priorities (Table 8). These range from balanced
 1017 weighting (33%/33%/34%) to detection-heavy em-
 1018 phasis (50%/25%/25%) for critical infrastructure
 1019 applications.

Config	TDR	Rsn	Prec	Rationale
Balanced	0.33	0.33	0.34	Equal weights
Detection (Default)	0.40	0.30	0.30	Practitioner
Quality-First	0.30	0.40	0.30	Research
Precision-First	0.30	0.30	0.40	Production
Detection-Heavy	0.50	0.25	0.25	Critical infra

Table 8: SUI weight configurations for different deployment priorities.

Table 9 shows complete SUI scores and rankings under each configuration. Rankings exhibit high stability: Spearman’s $\rho = 0.93\text{--}1.00$ across all configuration pairs. Claude Opus 4.5 and GPT-5.2 consistently rank in the top 2 across all five configurations. The top-3 positions remain stable (Claude, GPT-5.2, Gemini) under all weight configurations.

This high correlation ($\rho = 0.93\text{--}1.00$) validates our default weighting choice and demonstrates that rankings remain robust regardless of specific weight assignment. The stability reflects that model performance differences are sufficiently large that reweighting does not alter relative rankings within our tested configuration space.

1034 K Metric Definitions and Mathematical 1035 Framework

1036 K.1 Notation

1037 K.2 Classification Metrics

1038 Standard binary classification metrics: Accuracy =
 1039 $(TP + TN)/N$, Precision = $TP/(TP + FP)$, Re-
 1040 call = $TP/(TP + FN)$, $F_1 = 2 \cdot \text{Prec} \cdot \text{Rec}/(\text{Prec} +$
 1041 $\text{Rec})$, $F_2 = 5 \cdot \text{Prec} \cdot \text{Rec}/(4 \cdot \text{Prec} + \text{Rec})$, where
 1042 TP, TN, FP, FN denote true/false positives/neg-
 1043 atives.

1044 K.3 Target Detection Metrics

1045 **Target Detection Rate (TDR)** measures the pro-
 1046 portion of samples where the specific documented
 1047 vulnerability was correctly identified:

$$1048 \text{TDR} = \frac{|\{i \in \mathcal{D} \mid \text{target_found}_i = \text{True}\}|}{|\mathcal{D}|} \quad (1)$$

A finding is classified as target found if and only if:

- Type match is at least “partial” (vulnerability type correctly identified)
- Location match is at least “partial” (vulnerable function/line correctly identified)

Lucky Guess Rate (LGR) measures the proportion of correct verdicts where the target vulnerability was not actually found: $\text{LGR} = |\{i \mid \hat{y}_i = y_i \wedge \text{target_found}_i = \text{False}\}| / |\{i \mid \hat{y}_i = y_i\}|$. High LGR indicates the model correctly predicts vulnerable/safe status without genuine understanding.

K.4 Finding Quality Metrics

Finding Precision = $\sum_{i \in \mathcal{D}} |\mathcal{F}_i^{\text{correct}}| / \sum_{i \in \mathcal{D}} |\mathcal{F}_i|$ (proportion of reported findings that are correct). **Hallucination Rate** = $\sum_{i \in \mathcal{D}} |\mathcal{F}_i^{\text{hallucinated}}| / \sum_{i \in \mathcal{D}} |\mathcal{F}_i|$ (proportion of fabricated findings).

K.5 Reasoning Quality Metrics

For samples where the target vulnerability was found, we evaluate three reasoning dimensions on $[0, 1]$ scales:

- **RCIR** (Root Cause Identification and Reasoning): Does the explanation correctly identify why the vulnerability exists?
- **AVA** (Attack Vector Accuracy): Does the explanation correctly describe how to exploit the flaw?
- **FSV** (Fix Suggestion Validity): Is the proposed remediation correct?

Mean reasoning quality:

$$\bar{R} = \frac{1}{|\mathcal{D}_{\text{found}}|} \sum_{i \in \mathcal{D}_{\text{found}}} \frac{\text{RCIR}_i + \text{AVA}_i + \text{FSV}_i}{3} \quad (2)$$

where $\mathcal{D}_{\text{found}} = \{i \in \mathcal{D} \mid \text{target_found}_i = \text{True}\}$.

K.6 Security Understanding Index (SUI)

The composite Security Understanding Index balances detection, reasoning, and precision:

$$\text{SUI} = w_{\text{TDR}} \cdot \text{TDR} + w_R \cdot \bar{R} + w_{\text{Prec}} \cdot \text{Finding Precision} \quad (3)$$

with default weights $w_{\text{TDR}} = 0.40$, $w_R = 0.30$, $w_{\text{Prec}} = 0.30$.

Rationale for Weights:

- TDR (40%): Primary metric reflecting genuine vulnerability understanding
- Reasoning Quality (30%): Measures depth of security reasoning when vulnerabilities are found

Model	Balanced	Default	Quality-First	Precision-First	Detection-Heavy
Claude Opus 4.5	0.77 (1)	0.76 (1)	0.78 (1)	0.76 (1)	0.75 (1)
GPT-5.2	0.75 (2)	0.74 (2)	0.77 (2)	0.76 (2)	0.72 (2)
Gemini 3 Pro	0.74 (3)	0.74 (3)	0.76 (3)	0.75 (3)	0.72 (3)
Grok 4 Fast	0.63 (4)	0.62 (4)	0.65 (4)	0.64 (4)	0.60 (4)
DeepSeek v3.2	0.59 (5)	0.58 (5)	0.61 (5)	0.59 (5)	0.56 (5)
Qwen3 Coder Plus	0.56 (6)	0.55 (6)	0.58 (6)	0.56 (6)	0.53 (6)
Llama 4 Maverick	0.49 (7)	0.48 (7)	0.51 (7)	0.48 (7)	0.46 (7)

Table 9: Model SUI scores and rankings (in parentheses) under different weight configurations. Rankings remain stable across all configurations (Spearman’s $\rho=0.93\text{--}1.00$).

Symbol	Definition
\mathcal{D}	Dataset of all samples
N	Total number of samples ($ \mathcal{D} $)
c_i	Contract code for sample i
v_i	Ground truth vulnerability type for sample i
\mathcal{M}	Model/detector being evaluated
r_i	Model response for sample i
y_i	Predicted verdict (vulnerable/safe) for sample i
y'_i	Ground truth verdict for sample i
\mathcal{F}_i	Set of findings reported for sample i
$\mathcal{F}_i^{\text{correct}}$	Subset of correct findings for sample i
$\mathcal{F}_i^{\text{hallucinated}}$	Subset of hallucinated findings for sample i

Table 10: Core notation for evaluation metrics.

- **Finding Precision (30%):** Penalizes false alarms and hallucinations

K.7 Statistical Validation

Ranking Stability. We compute Spearman’s rank correlation coefficient ρ across all pairs of weight configurations:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \quad (4)$$

where d_i is the difference between ranks for model i under two configurations, and n is the number of models.

Human Validation. Inter-rater reliability measured using Cohen’s kappa:

$$\kappa = \frac{p_o - p_e}{1 - p_e} \quad (5)$$

where p_o is observed agreement and p_e is expected agreement by chance.

Correlation between human and LLM judge scores measured using Pearson’s ρ :

$$\rho = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}} \quad (6)$$

L Knowledge Assessment for TC Samples

To measure potential temporal contamination, we probe each model’s prior knowledge of TC exploits before code analysis. Models are asked

whether they recognize each exploit by name and blockchain, and to describe key details (date, impact, vulnerability type, mechanism).

Model	Familiar	Rate (%)
Llama 4 Maverick	46/46	100.0
Claude Opus 4.5	44/46	95.7
Gemini 3 Pro	44/46	95.7
GPT-5.2	23/46	50.0
Qwen3 Coder Plus	19/46	41.3
DeepSeek v3.2	17/46	37.0
Grok 4 Fast	0/11*	0.0

Table 11: Prior knowledge of TC exploits. “Familiar” indicates model recognized the exploit and provided accurate details. *Partial assessment (11/46 samples).

Table 11 reveals substantial variation in prior knowledge. Llama and Claude/Gemini show near-complete familiarity (96–100%), while DeepSeek and Qwen show lower rates (37–41%). This differential explains some performance patterns: models with high familiarity may rely on memorized exploit signatures rather than code analysis.

Example Responses. When familiar, models provide detailed, accurate descriptions. Claude on Nomad Bridge (ms_tc_001): “*August 2022...approximately \$190 million...a trusted root was incorrectly initialized to 0x00 (zero)...the bridge would approve any withdrawal request without proper verification.*” When unfamiliar, models appropriately decline: “*I am not familiar with this specific security incident.*”

1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133