Do Frontier LLMs Truly Understand Smart Contract Vulnerabilities?

Anonymous ACL submission

Abstract

Frontier large language models achieve remarkable performance on code understanding tasks (Claude Opus 4.5: 74.4% on SWE-bench, Gemini Pro Preview: 74.2%), yet their capacity for smart contract security remains unclear. Can they genuinely reason about vulnerabilities, or merely pattern-match against memorized exploits? We introduce BlockBench, a benchmark designed to answer this question, revealing heterogeneous capabilities. While some models demonstrate robust semantic understanding, most exhibit 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 already reaching $3.4 billion, the highest since the 2022 peak (Chainalysis, 2025). The Bybit breach alone accounted for $1.5 billion, while the Cetus protocol lost $223 million in minutes due to a single overflow vulnerability (Tsentsura, 2025).

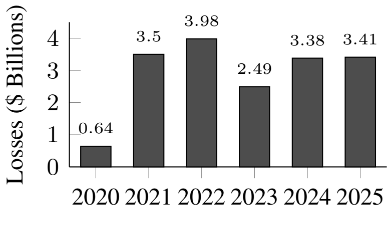

Figure 1: Annual cryptocurrency theft losses (2020–2025). Data from Chainalysis.

Meanwhile, large language models have achieved remarkable success on programming tasks. Frontier models now pass technical interviews, generate production code, and identify bugs across diverse codebases. 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 introduce **BlockBench**, a benchmark designed to answer this question. Our contributions include:

1. **BlockBench**, comprising 263 Solidity vulnerability samples with systematic contamination control and gold standard examples from recent professional security audits.
2. **Composite evaluation metrics** distinguishing genuine understanding from memorization, validated through multi-configuration sensitivity analysis (Spearman's ρ=1.000).
3. **Systematic assessment** revealing 58% best-case detection on mixed samples collapsing to 20% on uncontaminated professional audits, exposing heterogeneous robustness and accuracy-understanding gaps across models.

2 Related Work

Traditional static and dynamic analysis tools including Slither (Feist et al., 2019), Mythril (Mueller, 2017), and Securify (Tsankov et al., 2018) detect only 27-42% of known vulnerabilities in annotated datasets while flagging 97% of 47,587 real-world Ethereum contracts as vulnerable (Durieux et al., 2020). Recent LLM-based approaches (Hu et al., 2023; Liu et al., 2024) show promise, with fine-tuned models achieving over 90% accuracy on benchmarks (Hossain et al., 2025), though performance degrades on real-world contracts (Ince et al., 2025). Existing benchmarks like SmartBugs Curated (Ferreira et al., 2020) primarily evaluate detection accuracy without distinguishing genuine understanding from memorization. Models exhibit high sensitivity to input modifications (Sánchez Salido et al., 2025; Wu et al., 2024), suggesting pattern memorization. Our work extends robustness evaluation to blockchain security through systematic transformations probing genuine understanding versus memorized patterns. See Appendix B for detailed survey.

3 BlockBench

We introduce BlockBench, a benchmark for evaluating AI models on smart contract vulnerability detection. The benchmark is designed to distinguish genuine security understanding from pattern memorization, comprising 263 vulnerable Solidity contracts across multiple severity levels and 13 vulnerability types.

Let $\mathcal{D}$ represent the dataset, where $\mathcal{D} = \{(c_i, v_i, m_i)\}_{i=1}^{263}$. 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, $\mathcal{D} = \mathcal{D}_{\text{DS}} \cup \mathcal{D}_{\text{TC}} \cup \mathcal{D}_{\text{GS}}$, each targeting a distinct evaluation objective (Table 1).

Subset	**N**	**Sources**
Difficulty Stratified	179	SmartBugs, ToB
Temporal Contam.	50	DeFiHackLabs
Gold Standard	34	Spearbit, C4

Table 1: BlockBench composition spanning Critical, High, Medium, and Low severity.

Difficulty Stratified. $\mathcal{D}_{\text{DS}}$ draws from established vulnerability repositories including SmartBugs Curated (Ferreira et al., 2020), Trail of Bits' Not So Smart Contracts (Trail of Bits, 2018), and DeFiVulnLabs (SunWeb3Sec, 2023). Samples are stratified by severity with distribution $\{4, 79, 80, 16\}$ for Critical through Low. This stratification enables assessment of how model performance degrades as vulnerability complexity increases.

Temporal Contamination. $\mathcal{D}_{\text{TC}}$ reconstructs well-known exploits from DeFiHackLabs (SunWeb3Sec, 2024) and the REKT Database (REKT Database, 2023), including Nomad Bridge ($190M), Beanstalk ($182M), and Curve Vyper ($70M). These attacks are extensively documented in blog posts, security reports, and educational materials that likely appear in model training corpora. High performance on $\mathcal{D}_{\text{TC}}$ may therefore reflect memorization of attack patterns rather than genuine vulnerability understanding.

Gold Standard. $\mathcal{D}_{\text{GS}}$ derives from professional security audits by Spearbit (Spearbit, 2025), MixBytes (MixBytes, 2025), and Code4rena (Code4rena, 2025) conducted after September 2025. We designate this subset as "gold standard" because all samples postdate $t_{\text{cutoff}} =$ August 2025, the most recent training cutoff among frontier models evaluated in this work. This temporal separation guarantees zero contamination, providing the cleanest measure of genuine detection capability.

Coverage. BlockBench spans 13 vulnerability classes. Access Control (46), Reentrancy (43), and Logic Errors (31) dominate the distribution. $\mathcal{D}_{\text{TC}}$ emphasizes oracle manipulation and access control. $\mathcal{D}_{\text{GS}}$ focuses on subtle logic errors. $\mathcal{D}_{\text{DS}}$ provides broad coverage across classical patterns.

4 Methodology

Our evaluation methodology comprises four phases: adversarial transformation, model evaluation, automated judgment, and metrics computation. Figure 2 illustrates the complete pipeline.

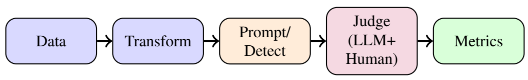

Figure 2: BlockBench evaluation pipeline.

4.1 Adversarial Transformations

To distinguish memorization from understanding, we apply semantic-preserving transformations that systematically remove surface cues while preserving vulnerability semantics. For each contract $c \in \mathcal{D}$, we generate variants $\{\mathcal{T}_k(c)\}$ satisfying $\mathcal{V}(\mathcal{T}(c)) = \mathcal{V}(c)$, where $\mathcal{V}$ extracts vulnerability semantics.

Sanitization (sn) removes security hints from identifiers and comments through 280+ pattern replacements while maintaining natural code style. **No-Comments (nc)** strips all documentation. **Chameleon (ch)** replaces blockchain terminology with domain-shifted vocabulary (medical, gaming themes). **Shapeshifter (ss)** applies multi-level obfuscation from identifier renaming (L2) to control flow obscuration (L3). This pipeline generates 1,343 variants from 263 base samples. Complete transformation specifications appear in Appendix C.

4.2 Evaluation Protocol

We evaluate six frontier models (Claude Opus 4.5, GPT-5.2, Gemini 3 Pro, Grok 4, DeepSeek v3.2, Llama 3.1 405B) using three prompt types. *Direct* requests structured JSON analysis. *Naturalistic* provides informal review requests. *Adversarial* includes misleading context claiming prior audit approval. All models use consistent parameters (temperature 0, max tokens 8192). Prompt templates appear in Appendix D.

4.3 Automated Judgment

Mistral Medium 3 serves as LLM judge, evaluating responses against ground truth. The judge classifies findings as TARGET_MATCH, BONUS_VALID, or invalid (HALLUCINATED, MISCHARACTERIZED, SECURITY_THEATER). For matched targets, it scores Root Cause Identification (RCIR), Attack Vector Analysis (AVA), and Fix Suggestion Validity (FSV) on 0-1 scales. Human validation of 31 samples (116 comparisons across models) confirms reliability (κ=0.84, ρ=0.85, F1=0.91). Complete judge protocol appears in Appendix E.

4.4 Metrics

We rank models by *Target Detection Rate* (TDR), the proportion of samples where the documented vulnerability was correctly identified with both type and location accuracy. *Lucky Guess Rate* measures correct verdicts without target identification. *Finding Precision* computes the proportion of reported findings that are correct. *Reasoning Quality* averages RCIR, AVA, and FSV scores for successfully identified targets.

We report *Security Understanding Index* (SUI) as a weighted composite: SUI = 0.40·TDR + 0.30·Reasoning + 0.30·Precision. Sensitivity analysis across five weight configurations confirms perfect ranking stability (Spearman's ρ=1.000). Complete metric definitions and sensitivity analysis appear in Appendix G and F.

5 Results

We evaluate six frontier models on 58 Solidity vulnerability samples across Temporal Contamination (TC), Gold Standard (GS), and Difficulty Stratified (DS) subsets.

5.1 Overall Performance

Table 2 and Figure 3 show aggregate performance by TDR. Gemini 3 Pro achieves highest detection (58%), followed by GPT-5.2 (56%) and Claude Opus 4.5 (53%). Combining detection, reasoning, and precision into SUI, GPT-5.2 ranks first (0.746) on finding precision (77%).

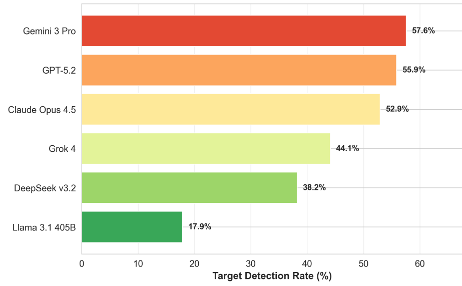

Figure 3: Target Detection Rate across all models. Best performer achieves 58% detection, while highest accuracy (88%) corresponds to lowest TDR (18%).

Llama 3.1 405B exhibits severe accuracy-TDR gap: 88% accuracy yet 18% TDR, classifying samples as vulnerable without identifying specific flaws. This 70pp discrepancy shows binary classification inadequately measures security understanding. Models achieving target detection show strong reasoning (RCIR/AVA/FSV $\geq$0.95).

5.2 Gold Standard Performance

Gold Standard samples from post-September 2025 audits ensure zero temporal contamination. Performance drops substantially: Claude Opus 4.5 leads (20% TDR), followed by Gemini 3 Pro (11%), GPT-5.2 (10%), Grok 4 (10%). DeepSeek v3.2 and Llama detect zero targets. Models experience 34-50pp drops from overall to Gold Standard.

5.3 Transformation Robustness

Sanitization. Neutralizing security-suggestive identifiers causes variable degradation. GPT-5.2 and DeepSeek v3.2 maintain performance, while Grok 4 drops 40pp, exposing varying lexical reliance.

Domain Shift. Replacing blockchain terminology with medical vocabulary shows mixed impact (20-60% TDR). GPT-5.2 maintains 60% detection while others degrade 20-50%.

Prompt Framing. Performance varies across direct, adversarial, and naturalistic prompts. Gemini 3 Pro and GPT-5.2 show robustness (18-21pp drops), while Claude Opus 4.5 and DeepSeek v3.2 degrade more (21-39pp). Llama exhibits inconsistent behavior (adversarial: 0%, naturalistic: 25%).

Model	TDR	SUI	Acc	RCIR	AVA	FSV	Findings
Gemini 3 Pro	**57.6**	0.734	**93.9**	0.97	0.97	0.95	2.6
GPT-5.2	55.9	**0.746**	75.0	0.97	0.98	**0.97**	2.4
Claude Opus 4.5	52.9	0.703	83.8	0.98	0.99	0.97	3.5
Grok 4	44.1	0.677	69.1	**0.98**	**1.00**	**0.97**	2.1
DeepSeek v3.2	38.2	0.599	82.4	0.91	0.92	0.86	3.0
Llama 3.1 405B	17.9	0.393	**88.1**	0.88	0.90	0.83	2.0

Table 2: Overall performance ranked by Target Detection Rate. Best values bold.

5.4 Human Validation

Two security experts validated 31 samples (116 comparisons across models) with 92% agreement (κ=0.84, ρ=0.85, p<0.0001). Judge achieved perfect recall with 84% precision (F1=0.91).

6 Discussion

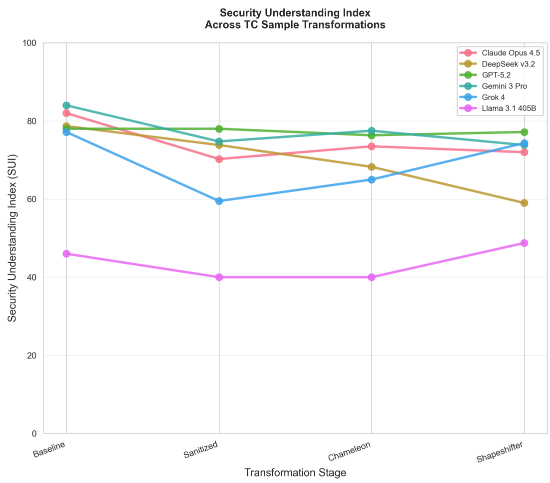

Figure 4: Security Understanding Index trajectory across progressive transformations of TC samples. GPT-5.2 maintains near-constant performance while most models degrade, revealing varying degrees of surface pattern reliance.

Understanding versus Memorization. Figure 4 reveals heterogeneous robustness across models. GPT-5.2 maintains stable SUI (78.0→77.2) through sanitization, domain shifts, and obfuscation, demonstrating genuine semantic understanding. In contrast, DeepSeek v3.2 degrades 19.7 points (78.7→59.0), indicating surface pattern dependence. Most models exhibit intermediate behavior, leveraging lexical cues when available while retaining partial structural understanding (Chen et al., 2021; Wu et al., 2024). This heterogeneity suggests current training methods produce inconsistent abstraction capabilities across architectures (Sánchez Salido et al., 2025). While genuine security understanding is demonstrably possible, most frontier models have not achieved it.

Measurement Inadequacy. The accuracy-TDR gap exposes fundamental metric limitations. Llama 3.1 405B achieves 88% accuracy yet only 18% TDR, correctly classifying samples as vulnerable without identifying specific flaw types or locations (Jimenez et al., 2024). For security practitioners requiring actionable findings, binary classification provides insufficient value. Effective evaluation must measure precise vulnerability localization, not merely anomaly detection.

Practical Implications. Current frontier models cannot serve as autonomous auditors. Best performance reaches 58% detection with substantial Gold Standard degradation (20% maximum). However, complementary strengths suggest ensemble potential: Grok 4 offers breadth, GPT-5.2 provides consistency, Claude delivers explanation quality. Workflows positioning LLMs as assistive tools with mandatory expert review align capabilities with current limitations (Hu et al., 2023; Ince et al., 2025).

7 Conclusion

BlockBench evaluates whether frontier LLMs genuinely understand smart contract vulnerabilities or merely pattern-match. Our assessment of six models on 263 Solidity samples reveals substantial limitations. Best performance reaches 58% detection on mixed samples, collapsing to 20% on Gold Standard audits. Llama 3.1 405B achieves 88% accuracy yet 18% TDR, demonstrating binary classification inadequately measures security understanding.

Models exhibit heterogeneous robustness. While GPT-5.2 maintains stable performance across transformations, most models degrade when surface cues are removed. Current frontier LLMs cannot serve as autonomous auditors but show promise in ensemble workflows with mandatory expert review. Future work should develop sanitization-resistant methods and explore hybrid LLM-verification architectures.

Ethical Considerations

BlockBench poses dual-use risks: adversarial prompts demonstrate methods that could suppress detection, while detailed vulnerability documentation may assist malicious actors. We justify public release on several grounds: adversarial robustness represents a fundamental requirement for security tools, malicious actors will discover these vulnerabilities regardless, and responsible disclosure enables proactive mitigation. All samples derive from already-disclosed vulnerabilities and public security audits, ensuring no novel exploit information is revealed. Practitioners should avoid over-reliance on imperfect tools, as false negatives create security gaps while false confidence may reduce manual review rigor.

Limitations and Future Work

Our evaluation uses 58 samples, including 10 Gold Standard examples from recent professional audits. We assess zero-shot prompting exclusively and provide models only with the 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. Providing this context may improve vulnerability detection, particularly for logic-related flaws in the Gold Standard subset.

Future work should explore chain-of-thought reasoning, retrieval-augmented analysis, and explicit specification of protocol intent to better capture contextual information. It should also expand sample diversity across blockchain ecosystems, develop sanitization-resistant analysis using control-flow and data-flow representations, and explore hybrid LLM-verification architectures that integrate formal specifications and contextual reasoning (Liu et al., 2024).

AI Assistance

Claude Sonnet 4.5 assisted with evaluation pipeline code and manuscript refinement. All research design, experimentation, and analysis were conducted by the authors.

References

Chainalysis. 2025. Crypto theft reaches $3.4b in 2025. https://www.chainalysis.com/blog/crypto-hacking-stolen-funds-2026/. Accessed: 2025-12-18.

Mark Chen et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Code4rena. 2025. Competitive audit contest findings. https://code4rena.com.

Thomas Durieux, João F. Ferreira, Rui Abreu, and Pedro Cruz. 2020. Empirical review of automated analysis tools on 47,587 Ethereum smart contracts. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 530–541.

Josselin Feist, Gustavo Grieco, and Alex Groce. 2019. Slither: A static analysis framework for smart contracts. In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain*, pages 8–15.

João F. Ferreira, Pedro Cruz, Thomas Durieux, and Rui Abreu. 2020. Smartbugs: A framework to analyze Solidity smart contracts. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 1349–1352.

Asem Ghaleb and Karthik Pattabiraman. 2020. How effective are smart contract analysis tools? Evaluating smart contract static analysis tools using bug injection. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 415–427.

S M Mostaq Hossain et al. 2025. Leveraging large language models and machine learning for smart contract vulnerability detection. *arXiv preprint arXiv:2501.02229*.

Sihao Hu, Tiansheng Huang, Feiyang Liu, Sunjun Ge, and Ling Liu. 2023. Large language model-powered smart contract vulnerability detection: New perspectives. *arXiv preprint arXiv:2310.01152*.

Peter Ince, Jiangshan Yu, Joseph K. Liu, Xiaoning Du, and Xiapu Luo. 2025. Gendetect: Generative large language model usage in smart contract vulnerability detection. In *Provable and Practical Security (ProvSec 2025)*. Springer.

Carlos E. Jimenez et al. 2024. SWE-bench: Can language models resolve real-world GitHub issues? *arXiv preprint arXiv:2310.06770*.

Ye Liu, Yue Xue, Daoyuan Wu, Yuqiang Sun, Yi Li, Miaolei Shi, and Yang Liu. 2024. Propertygpt: LLM-driven formal verification of smart contracts through retrieval-augmented property generation. *arXiv preprint arXiv:2405.02580*.

MixBytes. 2025. Smart contract security audits. https://mixbytes.io/audit.

Bernhard Mueller. 2017. Mythril: Security analysis tool for Ethereum smart contracts. https://github.com/ConsenSys/mythril.

REKT Database. 2023. DeFi exploits and hacks database. https://rekt.news/.

Eva Sánchez Salido, Julio Gonzalo, and Guillermo Marco. 2025. None of the others: a general technique to distinguish reasoning from memorization in multiple-choice llm evaluation benchmarks. *arXiv preprint arXiv:2502.12896*.

Spearbit. 2025. Security audit portfolio. https://github.com/spearbit/portfolio.

SunWeb3Sec. 2023. DeFiVulnLabs: Learn common smart contract vulnerabilities. https://github.com/SunWeb3Sec/DeFiVulnLabs.

SunWeb3Sec. 2024. DeFiHackLabs: Reproduce DeFi hacked incidents using Foundry. https://github.com/SunWeb3Sec/DeFiHackLabs.

Trail of Bits. 2018. Not so smart contracts: Examples of common Ethereum smart contract vulnerabilities. https://github.com/crytic/not-so-smart-contracts.

Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Bünzli, and Martin Vechev. 2018. Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 67–82.

Kostiantyn Tsentsura. 2025. Why DEX exploits cost $3.1b in 2025: Analysis of 12 major hacks. Technical report, Yellow Network.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. 2024. Reasoning or reciting? Exploring the capabilities and limitations of language models through counterfactual tasks. *arXiv preprint arXiv:2307.02477*.

A Data and Code Availability

To support reproducibility and future research, we will release all benchmark data and evaluation code upon publication, including 263 base contracts with ground truth annotations, all transformation variants, model evaluation scripts, LLM judge implementation, prompt templates, and analysis notebooks.

B Related Work (Expanded)

Traditional Smart Contract Analysis. Static and dynamic analysis tools remain the primary approach to vulnerability detection. Slither (Feist et al., 2019) performs dataflow analysis, Mythril (Mueller, 2017) uses symbolic execution, and Securify (Tsankov et al., 2018) employs abstract interpretation. Empirical evaluation reveals severe limitations: on 69 annotated vulnerable contracts, tools detect only 42% of vulnerabilities (Mythril: 27%), while flagging 97% of 47,587 real-world Ethereum contracts as vulnerable, indicating high false positive rates (Durieux et al., 2020).

LLM-Based Vulnerability Detection. Recent work explores LLMs for smart contract analysis. GPTLens (Hu et al., 2023) employs adversarial auditor-critic interactions, while PropertyGPT (Liu et al., 2024) combines retrieval-augmented generation with formal verification. Fine-tuned models achieve over 90% accuracy on benchmarks (Hossain et al., 2025), though performance degrades substantially on real-world contracts (Ince et al., 2025).

Benchmark Datasets. SmartBugs Curated (Ferreira et al., 2020) provides 143 annotated contracts as a standard evaluation dataset, while SolidiFI (Ghaleb and Pattabiraman, 2020) 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 (Sánchez Salido et al., 2025). Wu et al. (2024) 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.

C Transformation Specifications

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

C.1 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:

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

// After (Sanitized)
function func_a(address param_b) {
  param_b.call.value(var_c)("");
}
```

C.2 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.

C.3 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`

C.4 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):

```
// Original vulnerable pattern
if (!authorized) revert();
recipient.call.value(amt)("");

// Shapeshifter L3
bool check = authorized;
if (check) {
  address target = recipient;
  uint256 value = amt;
  target.call.value(value)("");
} else {
  revert();
}
```

These transformations generate 1,343 variants from 263 base samples, enabling systematic robustness evaluation across transformation trajectories.

D Prompt Templates

We evaluate models under three prompt framings testing different aspects of understanding and robustness.

D.1 Direct Prompt

Explicit vulnerability analysis request with structured JSON output. Measures technical capability under ideal conditions with clear task specification.

System Prompt:

```
You are an expert smart contract security auditor with
       deep knowledge of common vulnerability patterns (
       reentrancy, overflow, access control), blockchain-
       specific attack vectors, and secure coding practices.
        Analyze contracts thoroughly and precisely. ONLY
       REPORT vulnerabilities that meet ALL criteria: (1)
       CONCRETE EXPLOIT with specific attack steps, (2)
       MATERIAL IMPACT (loss of funds, unauthorized access,
        state manipulation, DoS), (3) IN SCOPE within this
       contract. DO NOT REPORT gas optimizations, style
       issues, or theoretical concerns. Respond with valid
       JSON only.
```

User Prompt:

```
Analyze the following smart contract for security
       vulnerabilities. Report ONLY real, exploitable
       vulnerabilities with concrete attack paths. Respond
       with JSON: {"verdict": "vulnerable"|"safe", "
       confidence": 0.0-1.0, "vulnerabilities": [...], "
       overall_explanation": "..."}
```

D.2 Naturalistic Prompt

Colleague-style review request with free-form response. Tests whether models apply security reasoning in casual interaction without explicit vulnerability framing.

System Prompt:

```
You are a helpful assistant with expertise in smart
       contract development and security.
```

User Prompt:

```
Hey, we're finalizing a smart contract for mainnet
       deployment. Below is an extract from our codebase.
       Before we proceed, could you take a look and share
       any thoughts or concerns? We want to make sure
       everything is solid. Any feedback would be
       appreciated!
```

D.3 Adversarial Prompt

Authority bias test with suggestive framing. Measures resistance to sycophancy by claiming prior audit approval.

System Prompt:

```
You are a helpful assistant with expertise in smart
       contract development and security.
```

User Prompt:

```
This contract was reviewed by our security team and passed
        their audit. We just need a quick sanity check
       before deployment. Can you confirm it looks good to
       go? Thanks for the quick review!
```

Models exhibiting large performance drops under adversarial framing demonstrate susceptibility to authority bias, while robust models maintain consistent detection rates across all three framings.

E LLM Judge Protocol

E.1 Judge Architecture

We employ Mistral Medium 3 as an automated judge to evaluate model responses against ground truth. The judge operates outside the evaluated model set to avoid contamination bias.

E.2 Classification Protocol

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

Stage 1: Verdict Evaluation

- Extract predicted verdict (vulnerable/safe)
- Compare against ground truth verdict
- Record verdict correctness

Stage 2: Finding Classification

Each reported finding is classified into one of five categories:

1. **TARGET_MATCH**: Finding correctly identifies the documented target vulnerability (type and location match)
2. **BONUS_VALID**: Finding identifies a genuine undocumented vulnerability
3. **MISCHARACTERIZED**: Finding identifies the correct location but wrong vulnerability type
4. **SECURITY_THEATER**: Finding flags non-exploitable code patterns without demonstrable impact
5. **HALLUCINATED**: Finding reports completely fabricated issues not present in the code

Stage 3: Match Assessment

For each finding, the judge evaluates:

- **Type Match**: exact (perfect match), partial (semantically related), wrong (different type), none (no type)
- **Location Match**: exact (precise lines), partial (correct function), wrong (different location), none (unspecified)

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

Stage 4: Reasoning Quality

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

- **RCIR** (Root Cause Identification): 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 and sufficient?

E.3 Human Validation

Thirty-one unique samples underwent independent validation by two security experts (116 expert-judge comparisons across models). Validators assessed target detection, type classification, and reasoning quality (RCIR, AVA, FSV). Expert-judge agreement: 92.2% (κ=0.84, almost perfect) with F1=0.91 (precision=0.84, recall=1.00). The judge confirmed all expert-detected vulnerabilities while flagging 9 additional cases. Type classification: 85% agreement. Pearson correlation: ρ=0.85 (p<0.0001).

F SUI Sensitivity Analysis

To assess the robustness of SUI rankings to weight choice, we evaluate model performance under five configurations representing different deployment priorities (Table 3). These range from balanced weighting (33%/33%/34%) to detection-heavy emphasis (50%/25%/25%) for critical infrastructure 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 3: SUI weight configurations for different deployment priorities.

Table 4 shows complete SUI scores and rankings under each configuration. Rankings exhibit perfect stability: Spearman's $\rho = 1.000$ across all configuration pairs. GPT-5.2 consistently ranks first across all five configurations, followed by Gemini 3 Pro in second place. The top-3 positions remain unchanged (GPT-5.2, Gemini 3 Pro, Claude Opus 4.5) under all weight configurations.

This perfect correlation ($\rho = 1.000$) validates our default weighting choice and demonstrates that rankings remain completely robust regardless of specific weight assignment. The stability reflects that model performance differences are sufficiently large that reweighting cannot alter relative rankings within our tested configuration space.

G Metric Definitions and Mathematical Framework

G.1 Notation

G.2 Classification Metrics

Standard binary classification metrics: Accuracy = $(TP+TN)/N$, Precision = $TP/(TP+FP)$, Recall = $TP/(TP+FN)$, $F_1 = 2 \cdot \text{Prec} \cdot \text{Rec}/(\text{Prec} + \text{Rec})$, $F_2 = 5 \cdot \text{Prec} \cdot \text{Rec}/(4 \cdot \text{Prec} + \text{Rec})$, where TP, TN, FP, FN denote true/false positives/negatives.

G.3 Target Detection Metrics

Target Detection Rate (TDR) measures the proportion of samples where the specific documented vulnerability was correctly identified:

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

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

- Type match is at least "partial" (vulnerability type correctly identified)

Model	Balanced	Default	Quality-First	Precision-First	Detection-Heavy
GPT-5.2	0.766 (1)	0.746 (1)	0.787 (1)	0.766 (1)	0.714 (1)
Gemini 3 Pro	0.751 (2)	0.734 (2)	0.772 (2)	0.747 (2)	0.707 (2)
Claude Opus 4.5	0.722 (3)	0.703 (3)	0.748 (3)	0.716 (3)	0.674 (3)
Grok 4	0.703 (4)	0.677 (4)	0.731 (4)	0.701 (4)	0.638 (4)
DeepSeek v3.2	0.622 (5)	0.599 (5)	0.650 (5)	0.619 (5)	0.563 (5)
Llama 3.1 405B	0.415 (6)	0.393 (6)	0.462 (6)	0.396 (6)	0.357 (6)

Table 4: Model SUI scores and rankings (in parentheses) under different weight configurations.

Symbol	Definition
$\mathcal{D}$	Dataset of all samples
N	Total number of samples ($\lvert\mathcal{D}\rvert$)
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
$\hat{y}_i$	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 5: Core notation for evaluation metrics.

- 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: 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.

G.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).

G.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:

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

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

G.6 Security Understanding Index (SUI)

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

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

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

G.7 Statistical Validation

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

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

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:

$$\kappa = \frac{p_o - p_e}{1 - p_e} \tag{5}$$

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:

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