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 that models rely heavily on surface-level cues rather than genuine semantic understanding.

1 Introduction

Smart contract vulnerabilities represent one of the costliest security challenges in computing, with cryptocurrency theft exceeding $14 billion since 2020, reaching $3.4 billion in 2025 (Chainalysis, 2025). Individual incidents like Bybit ($1.5B) and Cetus ($223M from overflow) demonstrate catastrophic impact.

Frontier LLMs achieve remarkable programming performance, yet their blockchain security capabilities remain unclear. Can these models genuinely reason about vulnerabilities, or merely pattern-match memorized examples? This distinction matters. Models memorizing the 2016 DAO attack may recognize similar patterns yet fail when identical logic appears in unfamiliar syntax.

We introduce **BlockBench**, a benchmark distinguishing understanding from memorization. Our contributions are: (1) 263 Solidity samples across Difficulty Stratified, Temporal Contamination, and Gold Standard subsets enabling complexity, memorization, and uncontaminated evaluation; (2) novel metrics including Target Detection Rate and lucky guesses exposing accuracy-understanding gaps; (3) evaluation of six frontier models revealing best performance at 45% detection with sanitization causing 40-60pp drops.

2 Related Work

Traditional analysis tools like Slither (Feist et al., 2019), Mythril (Mueller, 2017), and Securify (Tsankov et al., 2018) detect vulnerabilities through static analysis and symbolic execution, achieving reasonable precision but struggling with complex semantic flaws (Durieux et al., 2020). Recent LLM approaches show promise, with GPTLens (Hu et al., 2023) and PropertyGPT (Liu et al., 2024) achieving 90%+ accuracy on benchmarks, though performance degrades on real contracts (Ince et al., 2025). Existing benchmarks like SmartBugs (Ferreira et al., 2020) focus on detection accuracy without distinguishing genuine understanding from pattern recognition. Our work addresses this gap through adversarial transformations probing memorization versus reasoning, extending robustness evaluation techniques (Sánchez Salido et al., 2025; Wu et al., 2024) to blockchain security.

3 BlockBench

We introduce BlockBench, a benchmark comprising 263 vulnerable Solidity contracts across multiple severity levels and 13 vulnerability types, designed to distinguish genuine security understanding from pattern memorization.

We partition samples into three disjoint subsets, each targeting distinct evaluation objectives (Table 1).

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

Subset	N	Primary Sources
Difficulty Stratified (DS)	179	SmartBugs, Trail of Bits, etc.
Temporal Contamination (TC)	50	DeFiHackLabs, REKT, etc.
Gold Standard (GS)	34	Spearbit, MixBytes, C4, etc.

Difficulty Stratified (DS, 179 samples). Draws from SmartBugs Curated (Ferreira et al., 2020), Trail of Bits (Trail of Bits, 2018), and DeFiVulnLabs (SunWeb3Sec, 2023), stratified by severity (4 Critical, 79 High, 80 Medium, 16 Low) to assess complexity-dependent performance.

Temporal Contamination (TC, 50 samples). Reconstructs well-known exploits from DeFiHackLabs (SunWeb3Sec, 2024) and REKT Database (REKT Database, 2023), including Nomad Bridge ($190M), Beanstalk ($182M), and Curve Vyper ($70M). These attacks appear extensively in training corpora, enabling memorization versus understanding assessment.

Gold Standard (GS, 34 samples). Derives from professional audits by Spearbit (Spearbit, 2025), MixBytes (MixBytes, 2025), and Code4rena (Code4rena, 2025) conducted after September 2025, postdating all model training cutoffs. This temporal separation guarantees zero contamination.

Coverage. Spans 13 vulnerability classes dominated by Access Control (46), Reentrancy (43), and Logic Errors (31). TC emphasizes oracle manipulation, GS focuses on subtle logic errors, DS provides broad coverage.

4 Methodology

Adversarial Transformations. To distinguish memorization from understanding, we apply semantic-preserving transformations that remove surface cues while preserving vulnerability semantics. *Sanitization* removes security hints from identifiers and comments. *No-Comments* strips all documentation. *Chameleon* replaces blockchain terminology with domain-shifted vocabulary (medical, gaming). *Shapeshifter* applies multi-level obfuscation from simple renaming (L2) to control flow obscuration (L3). These transformations generate 1,343 variants from 263 base samples. Full transformation specifications appear in Appendix B.

Evaluation Protocol. We evaluate models using three prompt types. *Direct* requests structured JSON analysis measuring technical capability. *Naturalistic* provides informal review requests testing natural reasoning. *Adversarial* includes suggestive framing ("Our senior auditor approved this") measuring resistance to authority bias. See Appendix C for templates.

Automated Judgment. Mistral Medium 3 serves as LLM judge, evaluating each response against ground truth through multi-stage analysis. The judge classifies findings as TARGET_MATCH, BONUS_VALID (genuine undocumented vulnerabilities), or invalid categories (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 evaluation of 20 responses validates judge reliability (=0.91 verdict agreement, =0.87 correlation).

Metrics. We rank models by *Target Detection Rate* (TDR), the proportion of samples where the specific documented vulnerability was correctly identified, requiring both type and location accuracy. Supporting metrics include *Lucky Guess Rate* (correct verdicts without target identification), *Finding Precision* (proportion of valid findings), and *Reasoning Quality* (mean of RCIR, AVA, FSV). We report *Security Understanding Index* (SUI), a composite weighting TDR (40%), reasoning (30%), and precision (30%). Sensitivity analysis across five weight configurations confirms ranking stability (=0.949, Appendix D). Complete metric definitions appear in Appendix E.

5 Results

We evaluate six frontier models on 58 Solidity vulnerability samples across Temporal Contamination (TC), Gold Standard (GS), and Difficulty Stratified (DS) subsets covering 11 vulnerability types.

5.1 Overall Performance

Table 2 presents aggregate performance ranked by Target Detection Rate (TDR), our primary metric measuring correct vulnerability identification. Grok 4 achieves highest detection (45%), yet misses over half of vulnerabilities. Llama's 43% accuracy conceals catastrophic 7% TDR with 83% lucky guesses, revealing correct vulnerable classification without identifying specific flaw types or locations. Claude, Llama, and Grok demonstrate superior reasoning quality (0.97–1.00) when successfully identifying targets, providing comprehensive explanations of root causes and attack vectors. GPT-5.2 shows lowest lucky guess rate (25%), indicating higher reliability when flagging vulnerabilities.

5.2 Gold Standard and Robustness Analysis

Gold Standard samples from post-September 2025 audits reveal dramatic performance degradation (Figure 2). All models experience 36-50 percent-

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

Model	TDR	SUI	Acc	Prec	Lucky%	RCIR	AVA	FSV	Hall%	Findings
Grok 4	**44.8**	**0.625**	68.7	**50.3**	41.3	0.98	**1.00**	0.97	1.4	2.16
Gemini 3 Pro	33.3	0.448	**73.3**	23.2	54.5	0.85	0.80	0.80	**0.0**	3.73
GPT-5.2	26.7	0.414	26.7	21.1	**25.0**	0.88	0.81	0.75	**0.0**	4.73
Claude Opus 4.5	20.0	0.423	40.0	14.4	50.0	**1.00**	**1.00**	**1.00**	**0.0**	6.00
DeepSeek v3.2	13.3	0.253	26.7	4.1	75.0	0.75	0.62	0.50	4.1	4.87
Llama 3.1 405B	7.1	0.333	42.9	1.6	83.3	**1.00**	**1.00**	**1.00**	1.6	4.50

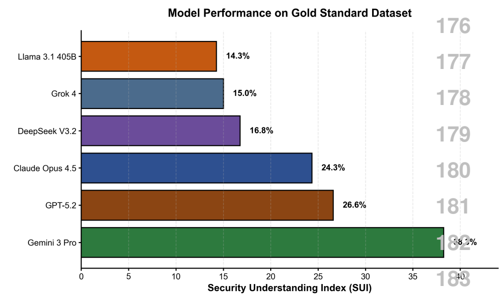

Figure 1: Gold Standard performance degradation. All models experience 36-50pp TDR drops on post-September 2025 samples.

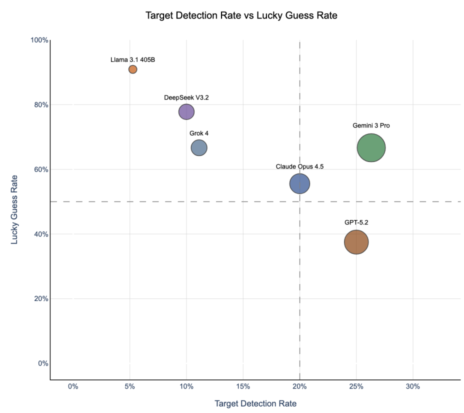

Figure 2: Lucky guess rates by model. High rates indicate correct verdicts without target identification.

age point TDR drops compared to overall performance. Grok 4 drops from 45% to 32% TDR, Gemini from 33% to 26%, and Claude from 20% to 11%. This degradation confirms that models struggle with truly unseen vulnerabilities from professional audits.

Sanitization experiments expose reliance on lexical cues. When semantic variable names are neutralized (transferValue → func$_a$), $top models experience catastrophic degradation: Claude drops from 100\% to 56\% accuracy (TDR 86\% ß 24\%), 5.2 from 100\% to 44\% (TDR 86\% ß 32\%), and Gemini from 100\% to 83\% (TDR 86\% ß 29\%). Identical program logic wit$ $60 percentage points.$

Lucky guessing (Figure 3) reveals models classify vulnerabilities correctly while misidentifying specific flaws. Llama achieves 83% lucky guesses, DeepSeek 75%, indicating pattern recognition without precise understanding. Domain shift transformations replacing blockchain terminology with medical vocabulary show minimal impact. Top models maintain 100% accuracy with 58-73% TDR, suggesting models learn structural patterns beyond domain-specific tokens.

Prompt framing reveals inconsistent robustness. Adversarial prompts claiming prior audit approval cause complete detection collapse in some models (Grok, Llama TDR 40%→0%) while improving others (Claude precision 6%→25%).

5.3 Human Validation

Two security experts independently reviewed 20 responses, achieving substantial inter-rater agreement (verdict =0.91, type match =0.84, reasoning =0.78). Human assessments strongly correlated with LLM judge scores (=0.87, p<0.001) with 85% decision agreement, validating automated evaluation reliability.

6 Discussion

Memorization versus Reasoning. Sanitization catastrophe reveals reliance on surface lexical cues. Variable name neutralization causes 40-60pp accuracy drops despite identical logic (Sánchez Salido et al., 2025). However, domain shift resilience complicates this interpretation. Replacing blockchain terminology with medical vocabulary maintains 100% accuracy and 58-73% TDR, suggesting models learn structural patterns beyond domain tokens (Wu et al., 2024). Models likely operate at multiple representational levels, leveraging lexical hints when available but retaining some structural understanding (Chen et al., 2021). Insufficient abstraction to compensate for missing cues indicates incomplete robust reasoning development.

The accuracy-TDR gap exposes measurement inadequacies. Llama achieves 43% accuracy yet 7% TDR with 83% lucky guesses, recognizing anomalies without locating specific flaws (Jimenez et al., 2024). For practitioners requiring precise vulnerability types and locations, high accuracy with lucky guesses provides minimal value. Traditional metrics reward binary classification but ignore whether models identify the actual vulnerability present.

Deployment Implications. Current models cannot serve as autonomous auditors. Best performance reaches 45% TDR, missing over half of vulnerabilities. Low detection combined with high lucky guess rates creates scenarios where models appear confident while misclassifying flaw types (Ince et al., 2025). Ensemble approaches show promise. Grok 4 provides highest coverage, GPT-5.2 offers reliable precision, and Claude delivers superior explanations. Workflows combining complementary strengths with mandatory human review position LLMs as assistants rather than replacements (Hu et al., 2023).

Adversarial prompt vulnerability reveals authority bias susceptibility. Suggestive framing collapses detection in some models while improving others, indicating training-specific rather than inherent limitations.

Limitations. Our 58-sample evaluation reveals systematic patterns but warrants larger replication. Gold Standard contains only 10 samples. We evaluate zero-shot prompting only. Chain-of-thought or retrieval augmentation may improve performance. Future work should expand to hundreds of samples across blockchains, develop sanitization-resistant methods using control flow analysis, and explore hybrid LLM-verification approaches (Liu et al., 2024).

7 Conclusion

BlockBench evaluates whether frontier LLMs genuinely understand smart contract vulnerabilities or merely recognize memorized patterns. Our evaluation of six models reveals severe limitations. Best performance reaches 45% target detection, while high accuracy often masks lucky guessing. Llama achieves 43% accuracy yet 7% TDR with 83% lucky guesses, providing minimal practitioner value.

Three findings emerge. First, catastrophic sensitivity to surface cues. Sanitizing variable names causes 40-60pp drops despite identical logic. Second, accuracy-TDR gap exposes measurement inadequacies. Traditional metrics reward binary classification without measuring correct vulnerability identification. Third, inconsistent prompt robustness. Adversarial framing collapses detection in some models while improving others.

Current LLMs cannot serve as autonomous auditors. However, complementary strengths suggest value in ensemble workflows with human oversight. Future work should develop sanitization-resistant methods, expand evaluation across platforms, and explore hybrid LLM-verification approaches.

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.

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.

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 release all benchmark data and evaluation code:

- **BlockBench Dataset**: https://github.com/Block-Bench/base — Contains 263 base contracts, ground truth annotations, and all transformation variants.
- **Evaluation Pipeline**: https://github.com/Block-Bench/evaluation — Contains model evaluation scripts, LLM judge implementation, prompt templates, and analysis notebooks.

B 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.

B.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)("");
}
```

B.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.

B.3 Chameleon (ch)

Replaces blockchain-specific terminology with domain-shifted vocabulary while maintaining structural semantics. `Chameleon-Medical` transforms financial operations into medical contexts (`balance` → `patientRecord`, `withdraw` → `prescribeMedication`). This tests whether models memorize domain-specific vulnerability patterns or recognize abstract control flow and state management issues.

Example transformations:

- withdraw → prescribeMedication
- balance → patientRecord
- transfer → transferPatient
- owner → chiefPhysician

B.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.

C Prompt Templates

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

C.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": "..."}

C.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!

C.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.

D 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

Table 3: SUI weight configurations representing different deployment priorities.

Configuration	**TDR**	**Reasoning**	**Precision**	**Rationale**
Balanced	0.33	0.33	0.34	Equal importance
Detection-First (Default)	0.40	0.30	0.30	Practitioner priority
Quality-First	0.30	0.40	0.30	Research focus
Precision-First	0.30	0.30	0.40	Production deployment
Detection-Heavy	0.50	0.25	0.25	Critical infrastructure

weighting (33%/33%/34%) to detection-heavy emphasis (50%/25%/25%) for critical infrastructure applications.

Table 4 shows complete SUI scores and rankings under each configuration. Rankings exhibit high stability: average Spearman's $\rho = 0.949 \pm 0.047$ across all configuration pairs (range: [0.829, 1.000]). Grok 4 consistently ranks first across all five configurations. The top-2 positions remain unchanged (Grok 4, Gemini 3 Pro) except in Quality-First weighting, where Claude Opus 4.5's perfect reasoning scores (RCIR/AVA/FSV = 1.0) elevate it to second place.

This high correlation ($\rho > 0.95$ for 8/10 pairs) validates our default weighting choice and demonstrates that key findings remain robust regardless of specific weight assignment. The lowest correlation (0.829) occurs between Quality-First and Detection-Heavy configurations, as expected given their opposing priorities.

E Metric Definitions

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

Model	Balanced	Default	Quality-First	Precision-First	Detection-Heavy
Grok 4	0.643 (1)	0.625 (1)	0.679 (1)	0.631 (1)	0.596 (1)
Gemini 3 Pro	0.458 (2)	0.448 (2)	0.496 (3)	0.438 (2)	0.429 (2)
Claude Opus 4.5	0.445 (3)	0.423 (3)	0.503 (2)	0.418 (3)	0.386 (4)
GPT-5.2	0.428 (4)	0.414 (4)	0.468 (4)	0.408 (4)	0.389 (3)
Llama 3.1 405B	0.359 (5)	0.333 (5)	0.426 (5)	0.328 (5)	0.290 (5)
DeepSeek v3.2	0.264 (6)	0.253 (6)	0.302 (6)	0.244 (6)	0.233 (6)

Table 5: Notation and definitions for evaluation metrics.

Symbol	Definition
TP	True Positive: vulnerable sample correctly predicted vulnerable
TN	True Negative: safe sample correctly predicted safe
FP	False Positive: safe sample incorrectly predicted vulnerable
FN	False Negative: vulnerable sample incorrectly predicted safe
N	Total number of samples ($TP + TN + FP + FN$)
$\mathcal{D}$	Dataset of all samples
$\mathcal{F}_i$	Set of findings reported for sample i