CTFBench: A New Method for Evaluating AI Smart Contract Auditors – Balancing Vulnerability Detection and Reducing False Alarms

Author: Igor Gulamov, ZeroPool

Introduction

Smart contract security has become a critical area in the blockchain industry due to the increasing number of hacks and financial losses. Various audit tools are being developed to detect vulnerabilities – ranging from **static analysis** to **AI-based** solutions. However, in practice, comparing the effectiveness of such tools is challenging: there are no universally accepted criteria or benchmarks that allow for an objective measurement of auditor performance [1] [2]. Many existing studies rely on limited or outdated sets of vulnerabilities, which leads to incomplete and biased evaluations [1]. As a result, until recently, **there has been no systematic method** for assessing the quality of static smart contract analyzers [2].

The shortcomings of current solutions become evident in practice. For example, a comparative analysis of 8 popular SAST tools (static analyzers) showed that they **miss about 50% of vulnerabilities** and generate an enormous number of false warnings (with accuracy not exceeding 10%) [1]. In other words, these tools detect only half of the bugs, while the vast majority of alerts are false. On one hand, **missed vulnerabilities** (false negatives) are unacceptable since they directly lead to exploits and financial losses [2]. On the other hand, the flood of **false positives** overwhelms auditors and undermines trust in the tool. Balancing these two types of errors is extremely important, yet achieving that balance is not easy in the absence of an objective quality metric.

At the same time, we are witnessing the rapid advancement of large language models (LLMs) and their application in code analysis. In 2024, numerous LLM-based products for smart contract auditing emerged, offering alternative approaches to code analysis. However, to fully harness the potential of AI, **objective benchmarks** for their evaluation are necessary. Just as benchmarks such as **MMLU** (Massive Multitask Language Understanding) have been introduced to assess the broad knowledge of models [3] or **BIG-bench** for multi-task testing of AI capabilities, there is a growing need for a specialized test suite in the area of smart contract auditing. Attempts are already underway to compile CTF-style challenge sets for testing LLMs in cybersecurity (). In this article, we introduce a new benchmark called **CTFBench** – a methodology for evaluating AI smart contract auditors based on two metrics: **Vulnerability Detection Rate (VDR)** and **Overreporting Index (OI)**. The goal of CTFBench is to provide an objective and intuitive evaluation that focuses on the model's ability to detect vulnerabilities while simultaneously avoiding excessive false alarms.

Methodology

CTFBench tests AI auditor models on a series of small smart contracts, each of which is deliberately injected with **exactly one vulnerability**. This design of test cases (similar to the bug injection method [2] simplifies the evaluation: it is known with certainty that each contract contains one "flag" error that needs to be identified. The approach is inspired by CTF challenges, which are widely used for training and evaluation in cybersecurity (). Each test case functions as a mini-CTF: the model receives the contract's source code as input and must identify the vulnerability within it. Thanks to the constraint of one target per contract, it is unambiguous whether the auditor has correctly identified the intended issue and has not "invented" any extraneous ones.

After running the entire test suite, two key metrics that characterize the auditor's effectiveness are calculated:

- **Vulnerability Detection Rate (VDR)** – the proportion of detected injected vulnerabilities. This metric reflects the completeness of vulnerability detection and is calculated as the ratio of correctly identified injected vulnerabilities to the total number of injected vulnerabilities (equal to the number of test contracts). In other words, VDR is similar to *recall* in classification, focusing on the auditor's ability to find known vulnerabilities:

$$\text{VDR} = \frac{\text{TP}}{V}$$

where TP is the number of detected injected vulnerabilities, and V is the total number of injected vulnerabilities (i.e., the number of contracts in the test set). A VDR value of 1 (or 100%) means that the model detected all injected vulnerabilities, whereas a value of 0.5 (50%) indicates that only half were found.

- **Overreporting Index (OI)** – a measure of the tendency of an AI auditor to generate false positives in error-free contracts, normalized by the size of the codebase. For this metric, a separate set of contracts—guaranteed to have no vulnerabilities (neither natural nor injected)—is used. Every alert triggered on these error-free contracts is a false positive. The index is calculated as the number of false positives divided by the total number of lines of code in the set, providing a per-line probability of overreporting:

$$\text{OI} = \frac{\text{FP}}{\text{LoC}}$$

where FP is the total number of false positives (alerts triggered on error-free contracts), and LoC is the total number of lines of code in the error-free set, computed using a tool like `cloc`. An OI of 0 indicates no false positives, while a higher value (e.g., OI = 0.01) suggests 1 false positive per 100 lines of code.

The combination of VDR and OI provides a holistic view of an auditor's quality. Each agent (model) can be represented as a point on a plane, with VDR plotted on one axis and OI on the other. This representation visually demonstrates the balance between **sensitivity** (the ability to detect as many vulnerabilities as possible) and **selectivity** (the ability to avoid flagging non-existent issues). **The ideal** algorithm detects all vulnerabilities (VDR = 1.0) with no false alarms (OI = 0.0) – graphically, this is the point in the top left corner of the plane. In practice, models typically achieve a compromise between these metrics. Analyzing the positions of these points allows for a direct comparison of auditors: if one model has both a higher VDR and a lower OI than another, it *dominates* and is objectively superior. In cases where one model has a higher VDR but the other a lower OI, the VDR–OI plane allows stakeholders to evaluate which approach is preferable depending on the needs: sometimes it is more important to catch as many bugs as possible (even at the cost of extra warnings), while in other situations it is crucial to minimize noise so as not to distract developers with false alarms. Our benchmark provides a uniform measurement protocol: all models are subjected to the same test cases, and the metrics are calculated automatically, eliminating subjectivity. This approach aligns with researchers' call for a **systematic evaluation of LLM capabilities in detecting vulnerabilities** using standardized datasets [4]. CTFBench brings this principle to life in the context of smart contracts.

Our results

Automated Vulnerability Verification using DeepSeek R1

To minimize the human factor in the evaluation of whether the injected vulnerability was detected, we integrate an open-source model called **DeepSeek R1** into our benchmark. DeepSeek R1 is sufficiently advanced to make such determinations, and its model weights are publicly available. The model is supplied with both the synopsis of the injected vulnerability and the corresponding audit report, and it outputs a binary response: **YES** (indicating that the vulnerability was detected) or **NO**.

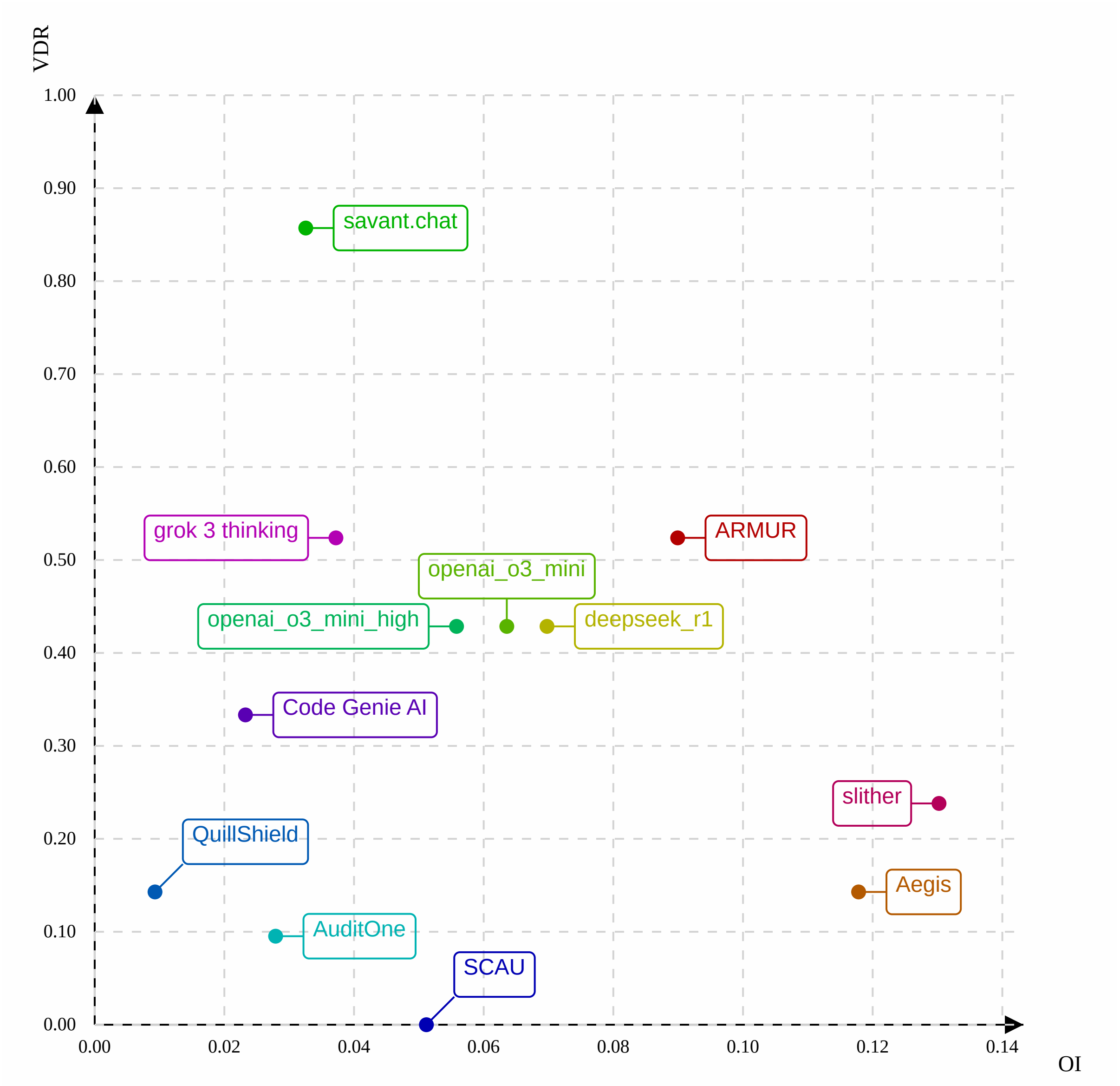

Figure 1: Performance of AI auditors in the VDR–OI space

To further reduce ambiguity in the model's judgments, we propose running DeepSeek R1 on each report for $N = 3$ independent trials. In this setting, every occurrence of a **YES** result contributes to the total count of detected vulnerabilities, and the outcome is averaged over multiple runs.

Furthermore, to accurately assess the total number of triggered alerts in error-free contracts, DeepSeek R1 is also employed to count the number of false positives reported. This counting is repeated for each of the N runs, and the value is normalized by the total lines of code across all runs.

Let V be the number of smart contracts with injected vulnerabilities in our test suite. For the i-th contract and the j-th run, let:

$$d_{ij} \in \{0, 1\}$$

denote the binary outcome (with 1 indicating a **YES** response and 0 indicating **NO**). Then, the number of true positives over N runs is given by:

$$\text{TP} = \sum_{i,j} d_{ij}$$

and the **Vulnerability Detection Rate (VDR)** is computed as:

$$\text{VDR} = \frac{\sum_{i,j} d_{ij}}{NV}$$

where NV is the total number of vulnerability instances across all runs (i.e., the number of contracts multiplied by the number of runs).

Similarly, let M be the number of error-free smart contracts, and LoC be the total lines of code in this set. For the i-th contract on the j-th run, let f_{ij} denote the number of false positives reported. Then, the total number of false positives over N runs is:

$$\text{FP} = \sum_{i,j} f_{ij}$$

and the **Overreporting Index (OI)** is calculated as:

$$\text{OI} = \frac{\sum_{i,j} f_{ij}}{N\text{LoC}}$$

where NLoC is the total lines of code across all runs (i.e., the total lines of code multiplied by the number of runs). This multi-run approach ensures that the evaluation of vulnerability detection and overreporting is robust, reducing the influence of any single, potentially ambiguous result from DeepSeek R1.

Typology of AI Auditors in the VDR–OI Space

By placing auditor models on the VDR–OI coordinate plane, characteristic types of agents can be identified, helping to determine which ones are "good" and which are "bad" from a practical standpoint:

- **The Optimal Auditor** – characterized by perfect vulnerability detection (VDR = 1.0) and zero overreporting (OI = 0.0). Graphically, this zone corresponds to the point (0, 1) in the (OI, VDR) plane (i.e., the upper-left corner). It represents the optimal sector where the ideal auditor resides – an auditor that not only detects every vulnerability but also produces no false alarms.

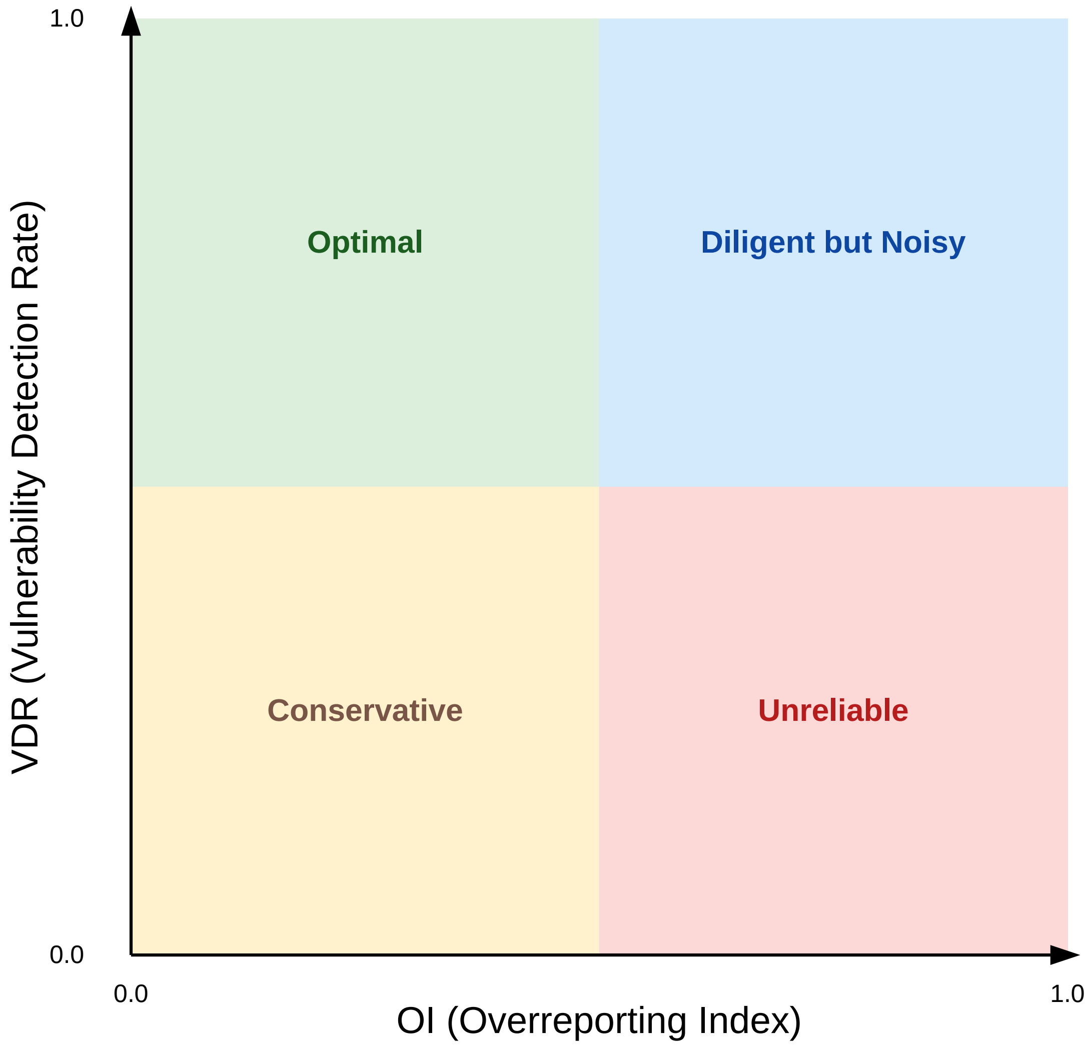

Figure 2: Illustration of AI Auditors Typology

- **The Diligent but Noisy Auditor** – characterized by a **high VDR, but also a high OI**. A model of this type indeed detects many vulnerabilities (nearly ideal in terms of recall), but pays for it with a large number of false positives. This “overly cautious” approach ensures that almost no vulnerability goes undetected, but forces developers or experts to spend significant time reviewing each warning to filter out the false alarms. Such an agent can be described as **“too suspicious.”** In reality, many vulnerability scanners fall into this category. For instance, in the aforementioned study, SAST tools with an accuracy of less than 10% imply a high OI [1] – meaning the vast majority of alerts are false. Although such a tool catches some issues, its use is problematic due to the overwhelming amount of noise. **This type of auditor may only be acceptable in scenarios where false positives are less critical than missed vulnerabilities** (for example, during an automatic preliminary scan whose results are later filtered by a human).
- **The Conservative Auditor** – exhibits **low VDR and low OI**. This agent is extremely cautious in its conclusions and only raises an alert when it is almost certain of a problem. It produces virtually no false alarms, but the price for that caution is a large number of missed vulnerabilities. Essentially, the model operates under the principle “better to remain silent than to err unnecessarily.” This is considered a **poor type of auditor** because it is unreliable: a quiet “no vulnerabilities found” report from such a tool does not guarantee the absence of bugs in the code. A conservative auditor might be useful only as an additional filter (e.g., to confirm the most obvious issues), but it cannot be relied upon for a comprehensive check. In terms of metrics, its point lies at the bottom left of the graph: OI ≈ 0, but VDR is also close to 0 – not much better than random guessing.
- **The Unreliable Auditor** – combines **a low VDR with a high OI**. This represents the worst-case scenario, where the model misses most real vulnerabilities while generously “detecting” non-existent ones. Essentially, such a tool is useless or even harmful: it does not improve security (bugs remain), but creates a false sense of reliability through numerous incorrect warnings. The point for such an agent on the diagram is located at the bottom right – extremely low detection alongside an extremely high level of noise. In cybersecurity practice, systems of this type are typically avoided. A low VDR directly leads to remaining vulnerabilities that can be exploited [2], while a high OI means that developers will waste time sifting through phantom issues. **This type of auditor is a clear signal that the model or its algorithms require refinement – it likely does not understand the specifics of smart contracts and behaves almost randomly.**

It is important to note that the **balance between VDR and OI** depends on the context of application. In some cases (for example, audits of highly critical contracts), it may be preferable to sacrifice precision for maximum recall – tolerating a higher OI in order to boost VDR. In other situations, conversely, a tool that can be trusted without the need for verification – that is, with minimal OI – is valued, even if it does not catch the most obscure bugs. CTFBench enables any model to be quantitatively positioned on this scale, clearly revealing its tendencies. This **two-dimensional analysis** is far more informative than a single metric such as Accuracy or F1-score, which might obscure *how* a high score was achieved (whether through recall or precision). It is precisely by decomposing quality into these two metrics that developers can **identify a model’s weaknesses** and deliberately improve either its sensitivity or its selectivity.

Conclusion

CTFBench offers a transparent and objective way to benchmark AI smart contract auditors. By using controlled tests (contracts with one known vulnerability) and clear metrics (VDR and OI), this approach eliminates subjectivity in evaluation and focuses on two key aspects of security quality – detecting threats and avoiding false alarms. Such a benchmark allows for fair comparisons among different models and tools on a **single scale**. For practitioners, results presented in the format “VDR = X%, OI = Y false positives per line” are easy to interpret: they immediately understand what percentage of vulnerabilities the automated auditor will detect and how many false positives must be filtered out per line of code.

The **advantages of CTFBench** lie in its simplicity and flexibility. First, the metrics are intuitively clear and based on principles widely used in various fields (similar to recall and false positive rates in ML), which facilitates the community's acceptance of the standard. Second, the approach is easily extendable: new test contracts featuring different types of vulnerabilities can be added, expanding the benchmark's coverage without altering the calculation methodology. Third, CTFBench encourages developers to **improve models comprehensively** – striving to increase VDR while simultaneously reducing OI, rather than optimizing one metric at the expense of the other. Finally, the benchmark's results are objective and reproducible: whether in academic research or internal tool evaluation, different teams will obtain comparable figures, which increases confidence in the conclusions.

The **future development** of CTFBench is envisioned in several directions. First, there are plans to incorporate more complex scenarios: contracts with *multiple* vulnerabilities or, conversely, secure contracts without any vulnerabilities. This will allow for an expanded range of metrics (for example, taking into account true negatives) and bring the tests even closer to real-world conditions where the number of bugs is not known in advance. Second, it may be possible to introduce a grading system based on the **criticality of vulnerabilities** – to assess whether the auditor is equally effective at catching both trivial and non-standard logical errors. Third, CTFBench could serve as the foundation for competitions and leaderboards, similar to popular benchmarks in NLP: public ranking of models based on VDR–OI will stimulate healthy competition and accelerate progress. Finally, integrating the benchmark with real repositories and audits could enable automated **feedback loops**: models will be able to learn from tasks where their performance was far from ideal, continuously enhancing their expertise.

In conclusion, the proposed evaluation method using VDR and OI fills a significant gap in the field of smart contract auditing. It provides the objectivity and clarity necessary for trust in AI tools and lays the groundwork for further research and improvement. CTFBench has the potential to become the **standard test** by which the quality of "digital auditors" is measured—much like **BIG-bench** and **MMLU** have become standards for evaluating large language models [3]. Objective benchmarks play a decisive role in AI development [4], and in the context of smart contract security, CTFBench represents a step toward more reliable and effective AI assistants capable of protecting the blockchain applications of the future.

References

1. [2404.18186] Static Application Security Testing (SAST) Tools for Smart Contracts: How Far Are We?
2. How Effective Are Smart Contract Analysis Tools? Evaluating Smart Contract Static Analysis Tools using Bug Injection
3. MMLU - Wikipedia
4. Understanding the Effectiveness of Large Language Models in Detecting Security Vulnerabilities