

Are state-of-the-art LLMs useful
for auditing or formal verification?

Kirill Balakhonov
Product @ Nethermind

Introduction



vitalik.eth 
@VitalikButerin

...

One application of AI that I am excited about is AI-assisted formal verification of code and bug finding.

Right now ethereum's biggest technical risk probably is bugs in code, and anything that could significantly change the game on that would be amazing.

Can LLMs find security issues in solidity smart contracts?

Potential concerns:

- Solidity is a niche language
- Vulnerabilities are complicated
- Findings are often arguable

State of the Field - 6 Months Ago

What was presented on the market regarding AI Smart Contract Auditing:

- Landing pages with AI Auditor that doesn't exist
- SaaS platforms with not implemented functionality to collect leads
- SaaS platforms which only run Slither (static analyzer)
- SaaS platforms which run ChatGPT and false positive findings
- Some research papers with conclusion we are not yet there.

Initial Experiments - Fine-tuning vs Closed Sourced

SB Curated: A Curated Dataset of Vulnerable Solidity Smart Contracts

SB Curated is a dataset for research in automated reasoning and testing of smart contracts written in Solidity, the primary language used in Ethereum. It was developed as part of the [execution framework SmartBugs](#), which allows the possibility to integrate tools easily, so that they can be automatically compared (and their results reproduced). To the best of our knowledge, SmartBugs Curated is the largest dataset of its kind.

Vulnerabilities

SmartBugs Curated provides a collection of vulnerable Solidity smart contracts organized according to the [DASP taxonomy](#):

Vulnerability	Description	Level
Reentrancy	Reentrant function calls make a contract to behave in an unexpected way	Solidity
Access Control	Failure to use function modifiers or use of tx.origin	Solidity
Arithmetic	Integer over/underflows	Solidity

Smart Bugs Curated Dataset Example

```
7  pragma solidity ^0.4.22;
8
9  contract FindThisHash {
10     bytes32 constant public hash = 0xb5b97fafd9855eec9b41f74df6c38f5951141f
11
12     constructor() public payable {} // load with ether
13
14     function solve(string solution) public {
15         // If you can find the pre image of the hash, receive 1000 ether
16         // <yes> <report> FRONT_RUNNING
17         require(hash == sha3(solution));
18         msg.sender.transfer(1000 ether);
19     }
20 }
```

Our fine-tuned LLM on HuggingFace

Models 17 Full-text search 11 Sort: Recently created

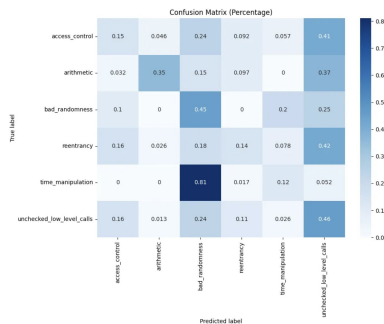
- PrunaAI/AlfredPros-CodeLlama-7b-Instruct-Solidity-bnb-4bit-smashed
Text Generation • Updated Aug 2 • 14
- andrijdavid/Solidity-Llama3-8b
Text Generation • Updated Jun 20 • 20
- hinny-coder/hinny-coder-6.7b-solidity-awq
Text Generation • Updated Apr 10 • 6
- hinny-coder/hinny-coder-6.7b-solidity-refine-awq
Text Generation • Updated Apr 10 • 5
- hinny-coder/hinny-coder-6.7b-solidity-final-awq
Text Generation • Updated Apr 25 • 4
- hinny-coder/hinny-coder-6.7b-solidity-refine
Text Generation • Updated Apr 9 • 10
- hinny-coder/hinny-coder-6.7b-solidity-final
Text Generation • Updated Apr 9 • 11
- hinny-coder/hinny-coder-6.7b-solidity
Text Generation • Updated Apr 9 • 12
- balakhonoff/solidity_security_model_merged_v2**
Text Generation • Updated Mar 25 • 4

Initial Experiments - Results

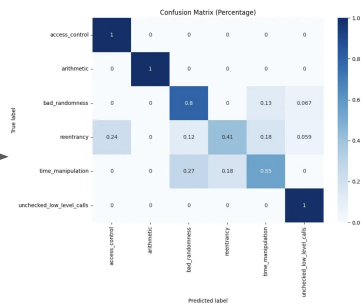
Vanilla GPT-4 drastically outperformed all fine-tuned LLMs (Mistral/LLaMa) on single-class detection task.

What is extremely important:

- The size of model (GPT-4 >> 13b..70b models)
- The annotation style (ask for line number)
- The detailed relevant context (context is key)



GPT-4o with some context



GPT-4o with relevant and comprehensive context

The prompt we used

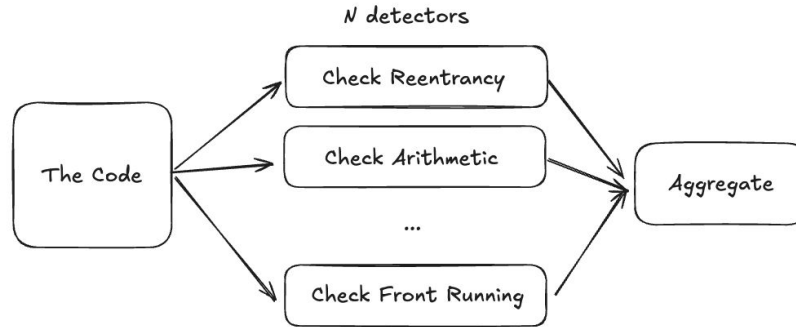
You are given a piece of Solidity code. Your task is to analyze this code for vulnerabilities of a specified type.
The type of vulnerability you need to look for is provided at the beginning of this prompt.
You must scan through the provided Solidity code and identify any instance(s) of this vulnerability.
Respond with the line number(s) at which each identified vulnerability instances. If multiple instances are found, separate the line numbers with a comma (e.g., "10,20,30"). Do not include whitespace between numbers.
If no instances of the specified vulnerability are found, respond with "None".
Vulnerability Type to Search For: ARITHMETIC which refers to issues arising from the way arithmetic operations (like addition, subtraction, multiplication, and division) are handled, potentially leading to overflow or underflow. This occurs when an operation results in a number exceeding the maximum or minimum size that can be stored within a variable's data type, altering the intended logic or value in a contract.

```
### Solidity code:
1: pragma solidity ^0.4.0;
2:
3: contract IntegerOverflowAdd {
4:     mapping (address => uint256) public balanceOf;
5:
6:
7:     function transfer(address _to, uint256 _value) public{
8:
9:         require(balanceOf[msg.sender] >= _value);
10:        balanceOf[msg.sender] -= _value;
11:        balanceOf[_to] += _value;
12:    }
13:
14: }
```

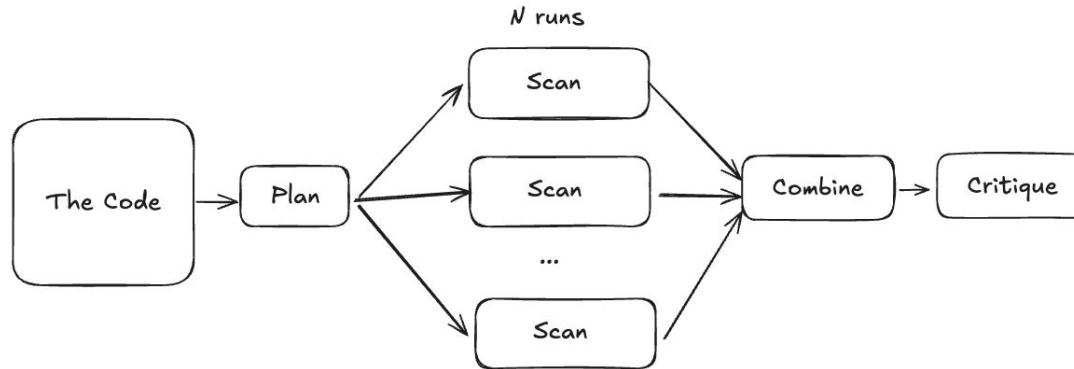
Vulnerable lines:
11,

Pivot to Agent-Based Approach

Before



After



The first positive response to the question "Are LLMs useful for auditing smart contracts?"

The finding by AI-agentic scanner:

Potential for unbounded loops in Registry contract

Contract(s) Affected: Registry

The removeContract function in the Registry contract uses a loop to find and remove a target contract. If the number of target contracts grows large, this could lead to high gas costs or even out-of-gas errors.

Recommendation: Implement a more gas-efficient data structure or mechanism for removing target contracts.

Developer feedback: *"Wait, this is important, actually I've never thought about it."*

Technical Evolution - OpenAI o1



Mudit Gupta ✓

@Mudit__Gupta

...

Has anyone tested the new OpenAI model for solidity audits?

It's surprisingly good from my initial testing. Good enough to become a mandatory step in all my audits.

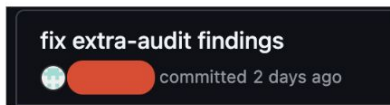
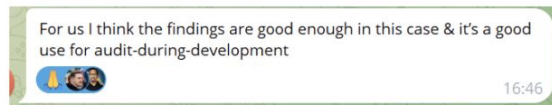
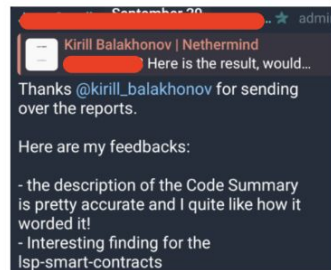
Definitely not at the level of a manual auditor but better than any static analyzer and automated audit tool.

Current State and Success Factors

In early October, this algorithm was able **to find a major bug in an already audited contract**, marking a new era in the use of LLM for auditing smart contracts.

The success of this approach comes down to four key factors:

1. Using LLMs with advanced reasoning capabilities (OpenAI o1-preview, o1-mini)
2. Implementing an agent approach with planned audit steps
3. Carefully selecting context for analysis runs, comprehensive prompts with examples of good and bad findings
4. Adding a quality critic & aggregation step




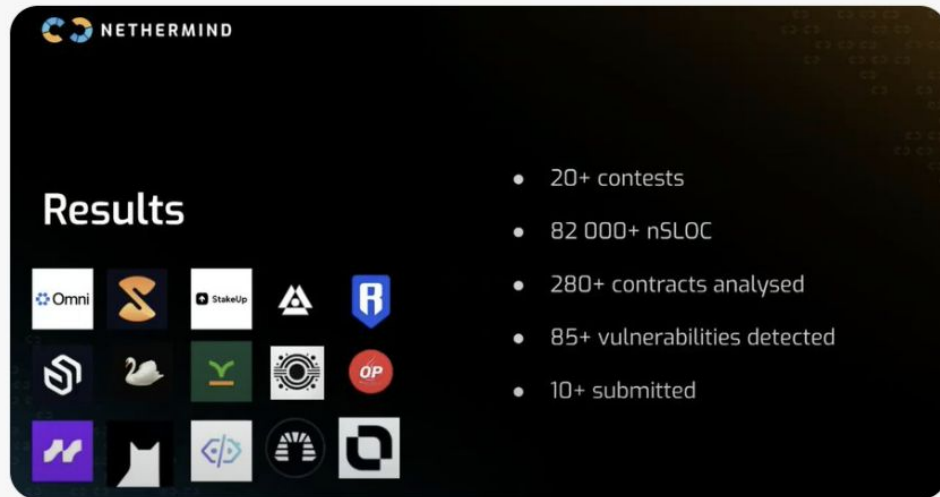


Kirill Balakhonov | Net  @balakr · 6 нояб.

[Начать продвижение](#)



 Exciting milestone to share: We've just completed an intensive 2-week security sprint with [@NethermindEth](#) & [@NethermindSec](#) AuditAgent, our AI-powered smart contract auditing tool. Here's what we achieved (in the thread below):



The graphic features the Nethermind logo at the top left. Below it, the word "Results" is prominently displayed. To the right of the title is a list of achievements. Below the title is a grid of 15 logos of various blockchain projects and organizations that participated in the sprint.

Results

- 20+ contests
- 82 000+ nSLOC
- 280+ contracts analysed
- 85+ vulnerabilities detected
- 10+ submitted

Logos included: Omni, StakeUp, R, and others.



3



22



59

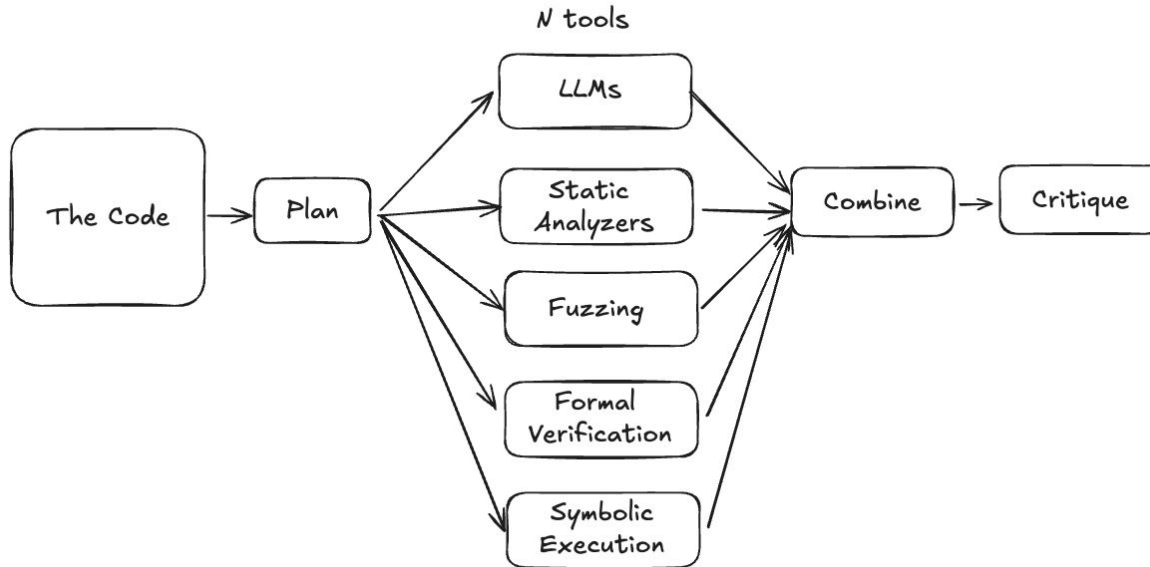


6 тыс.



Future Vision - SWE Agent Approach

The role of LLMs is not just to scan code, but also in **managing the tools that were previously available only to humans.**



Current Utility Matrix

What LLMs are useful for:

- ✅ Code auditing: Already finding bugs missed by human auditors
- ✅ Static analyzer use: Reducing false positives
- ↺ Fuzz testing: In development
- 💡 PoC writing: Showing promise but needs human oversight
- 💡 Formal Verification: Showing promise but needs human interaction
- 💡 Other Auditor Tools: Expecting this will be possible in the near future.

Questions?

Thank you for your attention.

I will be happy to answer any questions.

Twitter: @balakhonoff

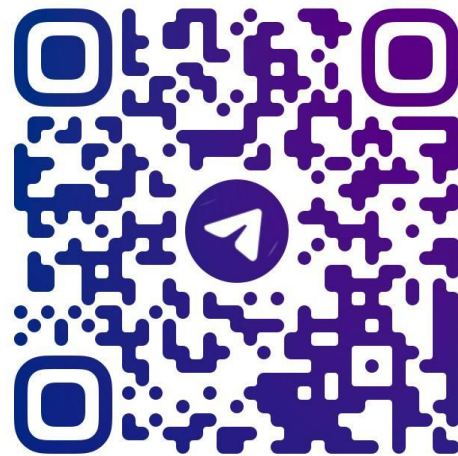
Telegram: @kirill_balakhonov

LinkedIn: /in/kirill-balakhonov

AuditAgent: AuditAgent.Nethermind.io

AUDITAGENT

AuditAgent.Nethermind.io



All links and a promo code for free access will be available in this group