TestMachine vs Large Language Models

How does TestMachine's performance compare to modern large language models?

December 11, 2023

Overview

Large language models (LLMs) such as ChatGPT-4 and Claude have revolutionized natural language processing but fundamentally lack an understanding of causality and sound reasoning. In domains like security analysis where logical rigor matters, Reinforcement Learning (RL) provides a compelling alternative to the use of LLMs. TestMachine, an RL-based smart contract penetration testing system, highlights the value of moving beyond the superficial text correlations that power LLMs.

By actively attempting hacks in a simulated environment, TestMachine's agents exposes incentives issues and state-based vulnerabilities that are beyond the capabilities of passive LLM code audits. The system unearths subtle attack vectors around manipulating voting, blocking withdrawals, or engineering insolvencies. TestMachine explicitly produces a sequence of actions — *e.g.*, a list of smart contract function calls — that will break a contract by draining tokens, violating access controls, or revealing private information. TestMachine then generates quantified metrics that allow developers to track risk reduction as they iterate on their codebase to harden their contracts and mitigate vulnerabilities.

This document discusses the well-known evidence that large language models frequently make invalid arguments and are prone to factual hallucinations, despite their prose fluency. We contrast the superficial reasoning of LLMs with TestMachine's mathematically principled exploration of a smart contract's state space. The fact that TestMachine executes actual exploits of smart contract vulnerabilities demonstrates reinforcement learning's advantages over LLMs' passive code analysis. As evidenced by dozens of successful hacks on sophisticated defi projects, TestMachine's focused algorithms mark a promising evolution in AI-based smart contract security.

The Rise of Large Language Models

Large language models are a class of natural language processing systems that have become possible recently thanks to advances in neural network architectures, computing power, and access to huge text datasets. They work by exposing a deep learning model called a transformer to vast corpora of natural language text data, allowing them to derive relationships between words and sentences based on staggering amounts of examples.

These models are trained using a technique called self-supervised learning. This means the model learns to predict the next word in a sequence by relying on the previous context, without requiring humans to manually label examples. Starting from simple sentences, the model gradually learns the probabilities of words co-occurring. As it processes more text — orders of magnitude more text and code than a human could possibly read in a lifetime — the model becomes extremely adept at constructing coherent sentences, filling in missing words, summarizing passages, translating languages, and generating code that mimics human output.

Large language models are so capable because they integrate details of language from a scope wider than any individual human could possibly be exposed to. They identify linguistic patterns across news articles, books, web pages and other text; grasp nuanced stylistic cues; incorporate cultural references and concepts; and benefit from the wisdom gleaned from nearly all recorded knowledge. While still inferior to humans in assessing truth or forming a coherent worldview, they excel at text-based tasks by distilling hard-won lessons from literally everything ever written.

Putting this power to use involves providing a "prompt" to focus the model's capabilities. The model generates predictive text continuations conditioned on the prompt, either creating original passages or completing patterns set by a human.

Limitations of Large Language Models

While large language models demonstrate an impressive grasp of linguistic knowledge and writing fluency, their underlying architectures struggle with complex reasoning that requires logical deduction or drawing inferences. Capable LLM models excel at continuing partial sentences, summarizing passages, or generating text around narrow prompts. *However, their completions necessarily follow statistical associations in their training data rather than adhering to principles of rationality or sound arguments.*

Several factors account for deficiencies in logical reasoning compared to humans. First, the self-supervised learning paradigm focuses exclusively on predicting the next token from preceding text. Without formal instruction in symbolic logic or exposure to structured reasoning tasks, models do not learn skills like assessing validity of arguments, spotting logical fallacies, or applying deductive thinking. Second, models lack a human-like memory, an understanding of physics, a representation of causality, or a grasp of abstract concepts needed for reasoning across varied contexts. Their knowledge is diffusely spread across neural network connection weights rather than as structured information.

Consequently, while often superficially persuasive, a language model's arguments often fail to withstand scrutiny. They regularly make assertions that contradict previous statements without awareness, hallucinate facts that don't exist, and make unjustifiable inferences around topics like ethics, science, or logic. Challenging their output reveals a fragility stemming from ungrounded statistical associations. More concerning from a security perspective, the flaws in their reasoning capabilities regularly lead to the algorithms missing critical vulnerabilities when evaluating smart contract code, even when specifically prompted to look for them.

TestMachine is Different

TestMachine utilizes reinforcement learning (RL) to actively reveal vulnerabilities in smart contracts before (and after) deployment. This contrasts with the large language models, which only passively analyze source code. In the RL approach, an agent interacts with contract logic in a simulated mainnet environment by constructing sequences of transactions. It receives rewards when exploits succeed in stealing tokens, locking access, *etc*. Over many iterations of acting, observing, and receiving feedback, the agent learns dynamic hacks that drain tokens, violate access controls, and expose private information in the contract — far exceeding what code audits by language models (or even humans, for that matter) can find.

For example, the RL agent can manipulate voting to sway decentralized governance, discover overflow bugs by carefully crafting inputs, or block withdrawals by front-running transactions. By actually executing attacks, the system reveals faulty assumptions, compiler issues, and incentives overlooked in static analysis.

The key difference from large language models is learning from experience and state changes rather than just static code structure. This allows developers to address logical gaps, prevent token heists, and institute fixes like circuit breakers. TestMachine also quantifies attack success rates to measure security improvement with each new contract version. By relentlessly exploiting contracts under safe conditions, TestMachine's RL penetration testing better prepares projects for adversarial environments.

In addition:

- Reinforcement learning dynamically interacts with contracts, rather than just passively analyzing code. This enables exploring and exploiting subtle state-based vulnerabilities that emerge through sequences of transactions.

- TestMachine can systematically test a wider range of malicious behavior like manipulating voting systems, front-running auctions, blocking withdrawals, *etc*. — going far beyond what static code audits by language models can find.
- Running attacks and probing edge cases on a mainnet simulation provides more realistic testing than just examining contract code in isolation, and can detect otherwise undetectable issues — *e.g.,* bugs in the Solidity or Vyper[1] compilers themselves.
- RL-based algorithms do not generate false positives — they either drain tokens, or they don't. And when they do find an exploit, TestMachine then leverages LLMs (an appropriate use case!) to provide developers detailed information on how the attack occurred, how to fix it, and even generate unit tests that implement the exploit.
- Feedback signals in RL (success/failure/rewards of attacks) drive focused learning on critical vulnerabilities rather than just coding best practices.
- Libraries of historical hacks, security research, and common anti-patterns can prime the RL approach to start at a sophisticated level before further iterations.
- Quantitative metrics on the hacker agent's success rate over time can benchmark progress as developers introduce fixes and improvements with each contract version.

TestMachine demonstrates the merits of task-specific reinforcement learning models over reasoning-limited large language models. By focusing advanced RL algorithms on the problem of smart contract security analysis, the system unlocks measurable security advancements that are unattainable from surface-level, LLM-based audit reviews.

[1] For example, TestMachine can reproduce the Curve Finance hack that exploited the Vyper compiler's faulty reentrancy locks; *cf.*, https://www.chainalysis.com/blog/curve-finance-liquidity-pool-hack/