Detecting Various DeFi Price Manipulations with LLM Reasoning

Juantao Zhong*, Daoyuan Wu*[1], Ye Liu†[2], Maoyi Xie‡, Yang Liu‡, Yi Li‡, and Ning Liu§
* Lingnan University, {ericzhong|daoyuanwu}@ln.edu.hk
† Singapore Management University, yeliu@smu.edu.sg
‡ Nanyang Technological University, maoyi001@e.ntu.edu.sg, {yangliu|yi_li}@ntu.edu.sg
§ City University of Hong Kong, City University of Hong Kong Shenzhen Research Institute, ninliu@cityu.edu.hk

***Abstract*—DeFi (Decentralized Finance) is one of the most important applications of today's cryptocurrencies and smart contracts. It manages hundreds of billions in Total Value Locked (TVL) on-chain, yet it remains susceptible to common DeFi price manipulation attacks. Despite state-of-the-art (SOTA) systems like DeFiRanger and DeFort, we found that they are less effective to non-standard price models in custom DeFi protocols, which account for 44.2% of the 95 DeFi price manipulation attacks reported over the past three years.**

In this paper, we introduce the first LLM-based approach, DeFiScope, for detecting DeFi price manipulation attacks in both standard and custom price models. Our insight is that large language models (LLMs) have certain intelligence to abstract price calculation from smart contract source code and infer the trend of token price changes based on the extracted price models. To further strengthen LLMs in this aspect, we leverage Foundry to synthesize on-chain data and use it to fine-tune a DeFi price-specific LLM. Together with the high-level DeFi operations recovered from low-level transaction data, DeFiScope detects various DeFi price manipulations according to systematically mined patterns. Experimental results show that DeFiScope achieves a high recall of 80% on real-world attacks, a precision of 96% on suspicious transactions, and zero false alarms on benign transactions, significantly outperforming SOTA approaches. Moreover, we evaluate DeFiScope's cost-effectiveness and demonstrate its practicality by helping our industry partner confirm 147 real-world price manipulation attacks, including discovering 81 previously unknown historical incidents.

I. Introduction

DeFi represents a form of finance that eliminates traditional intermediaries by utilizing smart contracts on a blockchain. A smart contract [1] is a self-executing program with the terms of the agreement directly written into code, deployed on a distributed, decentralized blockchain network [2]. These contracts power decentralized applications such as Automated Market Makers (AMMs), where token prices are determined by on-chain pricing mechanisms rather than centralized order books. While price manipulation is a pervasive risk in traditional finance, DeFi pricing mechanisms create more attack surfaces: adversaries can strategically add or remove liquidity, or impact token supply by minting or burning, to exploit vulnerabilities in pricing mechanisms, ultimately distorting prices of cryptocurrencies [3]. To detect price manipulation attacks in DeFi applications, researchers have proposed several state-of-the-art (SOTA) approaches, primarily focusing on transaction monitoring-based methods that offer real-time protection, e.g., DeFiRanger [4] and DeFort [5].

However, our analysis indicates that these SOTA approaches are less effective to non-standard price models in custom DeFi protocols. This is because they typically require token exchange rates to identify abnormal price changes, which are only suitable to calculate under standard price models such as CPMM (Constant Product Market Makers) and Stableswap Invariant (detailed in §II-A). Unfortunately, our subsequent evaluation shows that 44.2% of the 95 DeFi price manipulation attacks reported in the past three years used non-standard price models. Hence, *instead of explicitly calculating the exchange rate for a pair of tokens, we aim to capture only the abnormal price fluctuations of tokens*, which can be directly derived from the high-level price model and changes in token balances.

To achieve this objective, we sought help from large language models (LLMs), considering that their trained intelligence might aid in inferring price changes associated with DeFi operations during the transaction process. As such, we introduce DeFiScope, a source code-level detector and the first LLM-based approach to detecting DeFi price manipulation attacks in both standard and custom price models. DeFiScope features several novel designs, such as constructing a transfer graph to recover high-level DeFi operations (§V) and systematically mining price manipulation patterns across four prevalent types of DeFi applications (§VI). Among them, the key design is to fine-tune a DeFi price-specific LLM (§IV), where we propose (i) simulating transactions using Foundry [6] to generate our own fine-tuning data, and (ii) conducting a Chain-of-Thought (CoT)-style fine-tuning that integrates both on-chain data and the price context. The ablation study shows that fine-tuning increases the overall detection effectiveness by up to 31% compared to the baseline LLM (under DeFiScope) without fine-tuning.

To thoroughly evaluate DeFiScope's effectiveness and practicality, we conduct both a benchmark and a large-scale experiment. In the benchmark experiment, we collected 95 real-world price manipulation attacks from multiple sources and compared DeFiScope with three SOTA tools. The results show that DeFiScope achieves a superior detection accuracy (recall) of 80%, significantly higher than 51.6% with De-

1: Daoyuan Wu is the co-first author.
2: Ye Liu is the corresponding author.

FiRanger, 52.6% with DeFort, and 35.8% with DeFiTainter [7] (a static source code analysis tool). In our large-scale experiment, we evaluated DeFiScope on two datasets provided by our industry partner and DeFort [5]. The first dataset consists of 968 suspicious transactions, covering a variety of DeFi attacks beyond price manipulations. On this dataset, DeFiScope flagged 153 potential price manipulation attacks, 147 of which were manually verified as true positives, yielding a precision of 96%. Notably, 81 of these attacks are previously undocumented price manipulation incidents. The second dataset involves 96,800 benign transactions, on which DeFiScope produced zero false alarms. Regarding efficiency, DeFiScope incurs an average time overhead of 2.5 seconds per transaction across both suspicious and benign transactions, with a per-request LLM inference cost of only $0.0107.

In sum, this paper makes the following contributions:

- We introduce the first LLM-based approach, DeFiScope, for automatic on-chain price manipulation attack detection. In particular, LLMs, when properly fine-tuned, have the capability to abstract price models and infer price changes.
- To support DeFiScope's detection, we propose a graph-based method to recover high-level DeFi operations and systematically mine eight price manipulation patterns.
- We extensively evaluate DeFiScope with three real-world transaction datasets, showing DeFiScope's superior performance over SOTA tools in terms of precision and recall.

Artifact. To facilitate open science and future research, all source code, experimental results, and supplementary material are available in https://github.com/AIS2Lab/DeFiScope.

II. Background and Motivation

On blockchains, two primary account types exist, i.e., externally owned accounts (EOAs), controlled by individuals using private keys, and contract accounts (CAs), governed by their contract code. External transactions are initiated externally by EOAs, while internal transactions occurs when a smart contract calls another contract internally. Smart contracts exchange information with other smart contracts through internal transactions, where a transaction involving communication between contracts can be regarded as a sequence of function calls, utilized by DeFi protocols for interoperability. For simplicity, we denote as *user invocation* an internal transaction between the user-controlled smart contracts and other smart contracts, which plays a key role in price manipulation attacks.

A. Price Models

A price model represents the pricing mechanism within the DeFi application, which are typically expressed as equations correlating the price of a certain token with the balances of and the total supply of various tokens as well as other constant.

Constant Product Market Maker (CPMM). CPMM is one of the most prevalent DeFi AMMs and used in many well-known DEXs [8], [9]. It maintains liquidity through a constant function $x*y=k$, where k is a constant, and x and y represent the reserves of two distinct assets $token_x$ and $token_y$ in a liquidity pool. In a swap operation, let Δx amount of $token_x$ can exchange for Δy amount of $token_y$, resulting in $(x+\Delta x)*(y-\Delta y)=k$ where the instantaneous price of $token_x$ denominated in $token_y$ is $P_{x,y}=\frac{y}{x}$ [4], [10].

Stableswap Invariant. The Stableswap Invariant [11], widely used in Curve AMMs [12], and is defined as follows:

$$\frac{a\prod_{i=1}^{n}x_i}{(D/n)^n}\cdot D^{n-1}\cdot\sum_{i=1}^{n}x_i+\prod_{i=1}^{n}x_i=\frac{a\prod_{i=1}^{n}x_i}{(D/n)^n}\cdot D^n+\frac{D^n}{n^n}$$

where a indicates a constant amplification coefficient and n is the number of tokens in the liquidity pool, while D represents the total amount of tokens in the pool when token prices are equal, and x_i denotes the current reserve of $token_i$. The curve is nearly flat in the middle part, while price changes at the extremes resemble CPMM behavior.

Custom Price Model. Besides the mentioned common price models, DeFi protocols can customize their own pricing mechanisms, which are often more diverse and complicated. For example, UwULend [13] offers lending service, permitting users to borrow sUSDe by depositing cryptocurrencies as collateral. To ensure the user can repay, examining whether the total value of sUSDe are below that of the collateral when lending it is necessary. In this process, UwULend calculates the price of sUSDe by the following custom price model, distinct from the aforementioned CPMM and Stableswap Invariant models:

$$\begin{aligned}P_{sUSDe}=median(\{&IP_{USDe,Pool_1},\cdots,IP_{USDe,Pool_5},\\&EMAP_{USDe,Pool_1},\cdots,EMAP_{USDe,Pool_5}\})\end{aligned}\quad(1)$$

where P_{sUSDe} is the price of sUSDe, calculated as the median of a set of prices relevant to USDe in five liquidity pools[1]. $EMAP_{USDe,Pool_i}$ is the Exponential Moving Average (EMA) [19] price of USDe in the i-th liquidity pool but remains a constant value within a transaction block. In contrast, $IP_{USDe,Pool_i}$ refers to the instantaneous price of USDe in the i-th liquidity pool. Unlike the EMA, the latter price is more volatile and vulnerable to fluctuations in tokens' balance.

B. A Motivating Example

Our approach is motivated by a real-world price manipulation attack on UwULend [20], [21] in 2024. The root cause of the attack is the flawed price dependency in eq. (1) related to two functions `borrow` and `liquidationCall`. By swapping a large amount of USDe into five liquidity pools, the attacker deflated the instant price of USDe while the EMA prices kept unchanged, resulting in a lower median price for sUSDe. Hence, the attack was able to borrow an exceedingly large amount of sUSDe using the same amount of collateral. Then, swapping back could increase the calculated price of sUSDe, resulting in the attacker's collateral being unable to repay the debt, thereby allowing liquidation and acquiring deposited collateral with a bonus, leading to a $19M loss.

While detecting a price manipulation attack, existing tools generally define the exchange rate between two different tokens as their price. For example, the price of sUSDe could

[1] $Pool_{FRAXUSDe}$ [14], $Pool_{USDeUSDC}$ [15], $Pool_{USDeDAI}$ [16], $Pool_{USDecrvUSD}$ [17], and $Pool_{GHOUSDe}$ [18].

be calculated by dividing the amount of WETH deposited into the protocol by the amount of sUSDe borrowed from it. Yet, defining abnormal price change range or even precisely capturing token prices is not a trivial task.

- DeFort [5] uses historical exchange rates, i.e., prices, between two tokens to compute a so-called normal fluctuation range. However, it may be error-prone as historical prices, especially for low-liquidity or newly created pools, could vary significantly, thus being insensitive to the detection of subtle price manipulations. For instance, in the UwULend attack, slight manipulations observed in sUSDe prices — decreasing by 4.2 % or increasing by 4.43 %, did not exceed the predefined bounds that evades DeFort's detection, illustrated in our evaluation in §VII.
- DeFiRanger [4] detects abnormal price changes by tracing token exchange sequences within a transaction and comparing token exchange rates. However, completely tracing these sequences is challenging for complex transactions. In the UwULend attack, the attacker crafted complicated deposit and withdrawal operations concerning WETH, making it difficult for DeFiRanger to detect.

To address the aforementioned issues about price identification, our observation is that abnormal price fluctuations of tokens could be directly derived from the high-level price model and amount changes in token balances, without needing to explicitly calculate the exchange rate for a pair of tokens. Taking the UwULend attack as an example, swapping USDe into liquidity pools decreases the value of $IP_{USDe,Pool_i}$, causing an abnormal drop in the median price, namely the price of sUSDe. Based on this observation, in this paper, we propose a novel price change reasoning approach powered by LLM and integrate with predefined rules to enhance the capability of price manipulation attack detection.

III. Overview

Based on the analysis of the motivating example illustrated in §II-B, we establish an intuition of inferring price change associated with DeFi operations during the transaction process to detect DeFi price manipulations in various scenarios. While inferring through standard price models (i.e., CPMM and Stableswap as introduced in §II-A) is straightforward, it is challenging to (i) interpret low-level price calculation Solidity code into high-level price calculation formulas, and (ii) infer the price change from the (custom) price calculation formulas and the information on the token balance changes in related accounts. Both tasks require certain intelligence.

To address this key challenge, we introduce the first LLM-based approach, DeFiScope, for effective DeFi price manipulation detection. As depicted in Fig. 1, DeFiScope consists of ten steps. In step ①, DeFiScope first decodes and slices raw transaction data, and retrieves the source code of smart contracts called within the transaction. Then in steps ② ⑤ DeFiScope extracts the code of possible price calculation functions from smart contracts based on their signature, and token balance changes in relevant accounts. Subsequently, in steps ⑥, DeFiScope embeds these two information into a prompt template, which will be used by the LLM to extract the price model and infer the price change in steps ⑦ ⑧. In the meantime, in steps ③ ④, DeFiScope constructs the transfer graph (§V-A) and uses it to recover the high-level DeFi operations (§V-B) associated with those price change. This is because using the trend of token price changes alone is insufficient for detecting price manipulation. Finally, based on the recovered DeFi operations and their price change information, DeFiScope maps them into eight attack patterns listed in §VI and detects DeFi manipulation attacks in step ⑨.

We conduct an off-line fine-tuning step (step ⓪) to enhance LLMs' capabilities in extracting price calculation models and token price change reasoning. This is because while the off-the-shelf LLMs may exhibit certain capability in reasoning and code understanding, they are limited for predicting the trend of token price changes given unlabeled code snippet and numerical changes of tokens balance, which will be illustrated in §VII-B. We now detail our fine-tuning technique in §IV.

IV. Price Change Inference with LLMs

A. LLM Fine-tuning

For fine-tuning techniques, we *by default* use OpenAI's fine-tuning paradigm [22] instead of supervised fine-tuning (SFT) [23], [24], because the former requires only a small set of training data, whereas the latter generally requires more data points [25]. Nonetheless, we also demonstrate the *generalizability* of DeFiScope's fine-tuning with open-source SFT models; see details in §VIII. In the default setting, DeFiScope enhances the GPT-3.5-Turbo and GPT-4o models with data synthesized using the CPMM price model as illustrated in §II-A, along with on-chain data to fine-tune them.

Our fine-tuning process comprises two components: (a) on-chain data simulation, which generates realistic token balance-change pairs from transactions on a forked blockchain network; and (b) chain-of-thought (CoT)-style fine-tuning, where these simulated data are integrated into CoT-style prompts that are then used to fine-tune the LLM, enabling it to perform stepwise reasoning.

On-Chain Data Simulation. We leverage the fuzz testing method in Foundry [6], an off-the-shelf toolkit for Ethereum application development, to simulate on-chain data. To avoid data leakage and generate a substantial volume of transactions satisfying the CPMM, we select the Uniswap V2:BTC20 [26] liquidity pool as our target. We randomly generate inputs, namely integers ranging from 10^{20} to 10^{21} — 100 Ether to 1000 Ether, for the swap operations which are simulated on a forked blockchain of block height 17,949,214. Specifically, to include the data of inflating the price of tokens, we craft particular operations. To begin with, we record the balance of WETH and BTC20 in the liquidity pool denoted as bal_{WETH} and bal_{BTC20} respectively. Then we trigger `swapExactTokensForTokens` in contract `UniswapV2Router02` [27] to swap a amount of BTC20 for WETH, and record the latest balance of WETH, bal'_{WETH}, and that of BTC20, bal'_{BTC20}. Finally, we obtained the tokens' balance change as a pair (bal_{WETH} −

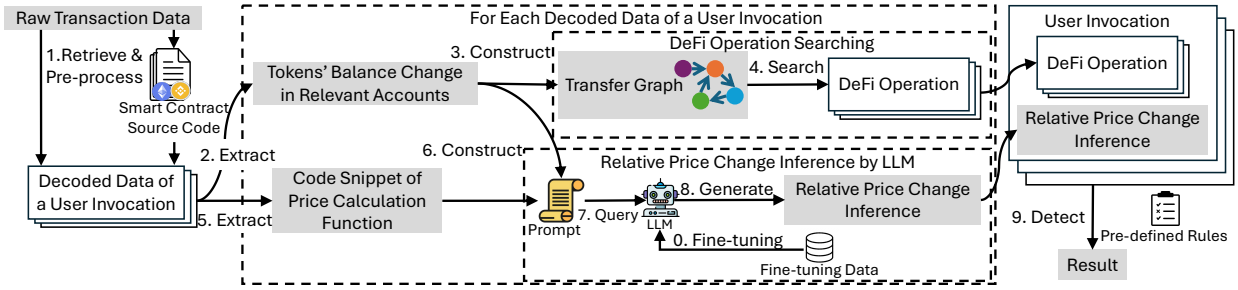


Fig. 1. A high-level overview of DeFiScope.

Simplified Fine-tuning Prompt Template

Instruction 1: [Instruction of extracting price model from code]
{code}

Instruction 2: [Instruction of scoring statement of price change tendency]
<statement>
1) **The price of input token increases after change**
2) **The price of input token decreases after change**
3) **The price of output token increases after change**
4) **The price of output token decreases after change**
</statement>
<change_description>
Balance of input token in the contract {direction of change} by {value$_0$}
Balance of output token in the contract {direction of change} by {value$_1$}
</change_description>
[Instruction and template of answer]
1) Score of **The price of input token increases after change**: {score}
2) Score of **The price of input token decreases after change**: {score}
3) Score of **The price of output token increases after change**: {score}
4) Score of **The price of output token decreases after change**: {score}

Fig. 2. The simplified prompt template used in fine-tuning the LLM. The full version can be found in our supplementary material [28].

$bal'_{WETH}, bal_{BTC20} - bal'_{BTC20}$). In terms of deflating the price of tokens, we swap a amount of WETH for BTC20 instead, with similar subsequent operations. Finally, we build a fine-tuning database with 500 pairs each for price inflating and deflating. Despite the only use of CPMM-based DeFi protocols, our evaluation results in §VII demonstrates a significant gain in term of price manipulation attack detection for DeFi protocols using custom price models.

CoT-style Fine-tuning. Fig. 2 demonstrates the prompt template used in the fine-tuning. We carefully construct a CoT-style fine-tuning prompt for guiding the LLM toward step-by-step reasoning, which integrates both simulated on-chain data collected in the previous step and the price context. Above the dashed line is the first instruction, which requires the LLM to extract the price calculation model from the provided code. {code} is the placeholder for the code snippet of price calculation functions. Below the dashed line, we guide the LLM to evaluate the credibility of four statements based on the price model extracted from step 1 and the tokens' balance change. We demand that the LLM expresses the credibility of a statement using integers ranging from 1 to 10. Compared to merely responding with a simple "Yes" or "No", this scoring method also indicates the confidence level of the responses, which can assist us in selecting the answers in which the LLM is more confident. By offering a finer-grained evaluation scale, it reveals when the model is uncertain—if two contradictory statements receive the same confidence score, we know it cannot distinguish which is more trustworthy and therefore discard those low-confidence results.

In the data part of the template, $\{value_0\}$ and $\{value_1\}$ are the placeholders for the first and second values in a price change pair, which are sampled from the fine-tuning dataset, respectively. {direction of change} can be either "increases" or "decreases." Specifically, if $\{value_i\}$ is greater than 0, {direction of change} is "increases"; conversely, if $\{value_i\}$ is less than 0, {direction of change} is "decreases." For the answer part, {score} is the score placeholder, an integer between 1 and 10.

Following OpenAI's fine-tuning guideline [22] that recommends using 50 to 100 training examples, we randomly sample 96 non-repetitive (in current and previous training sets) data from the fine-tuning database, and allocate 83% of the samples for training and 17% for validation. Subsequently, we insert data from the training and validation sets into the prompt template. To obtain the desired response for each prompt, we firstly ask LLMs to generate raw response, including the analysis of price model and scores of statements, for the given prompt. Next, we manually verify the correctness of the responses, of which the correct responses are stored and the wrong responses will also be corrected. Particularly, we simply swap the scores of two opposite statements to correct the error in them. Through this process, we construct a ground truth about prompts and its responses for fine-tuning.

During the fine-tuning process, training hyperparameters were automatically configured by the OpenAI fine-tuning API (for SFT fine-tuning, hyperparameters were manually set using common practice; see §VIII). In particular, once the model achieved 100% accuracy on the validation set, we terminated the training to avoid overfitting. The entire fine-tuning process is cost-efficient, consuming a total of around 1 million training tokens for each model. The detailed costs across different models will be introduced in §VII-B.

B. Inference with the Fine-tuned LLM

We first illustrate the *general inference process* using the motivating example. Nevertheless, there could be closed source DeFi protocols that dissatisfy steps ⑤⑥ so that we design a *customized inference process* to deal with them.

General Inference Process. To assist LLMs in extracting and analyzing the price model, our inference prompt provides relevant code for LLMs to score the trend of price changes and guides them in inferring the token price changes. Fig. 3 illustrates a simplified version of the prompt used and the response produced by our fine-tuned LLM

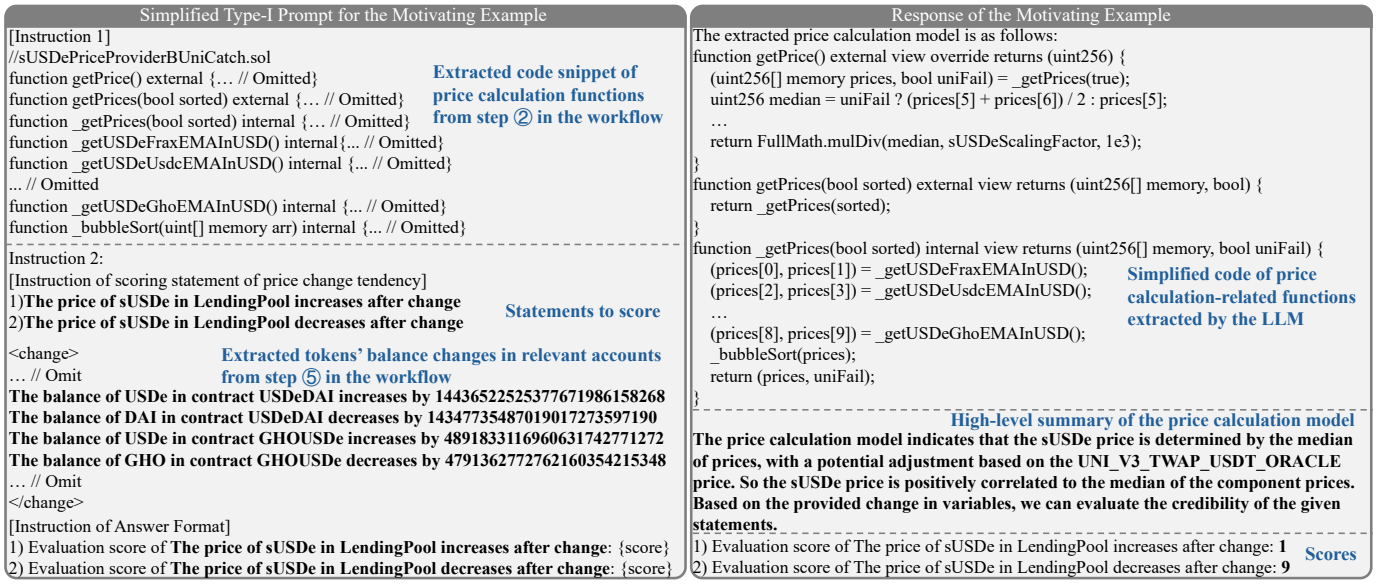


Fig. 3. The simplified Type-I prompt and its response for the motivating example during the inference process.

for inferring price changes of the motivating example in §II-B. We call this Type-I prompt, used for the typical cases with code input retrieved from steps ⑤⑥. It is different from the fine-tuning prompt in the `<statement>` and `<change_description>` parts. In the fine-tuning prompt shown in Fig. 2, both parts are fixed, while they are dynamically generated during inference using two formats: (i) "The price of {`token_name`} in {`contract_name`} {`direction_of_change`} after change" for the part `<statement>`, and (ii) "The balance of {`token_name`} in {`contract_name`} {`direction_of_change`} by {`change_value`}" or "The total supply of {`token_name`} {`direction_of_change`} by {`change_value`}" for the part `<change_description>`. Specifically, to fill them in Type-I prompt, DeFiScope generates a pair of statements for each token, i.e., one regarding the increase in token price and another regarding the decrease in token price.

DeFiScope asks the fine-tuned LLM to locate the price calculation model from the input code and evaluate the credibility of the generated statements. From the motivating example's response shown in the right-hand section of Fig. 3, the LLM first extracts the code of price calculation-related functions, followed by a high-level summary. In this example, it accurately identifies the underlying price model (see Equation (1)): the price of `sUSDe` is determined by the median of multiple prices. With this knowledge, the LLM can evaluate two opposing statements based on the changes in token balances and provide the correct answer with high confidence, i.e., a higher score.

Customized Inference Process. Although the majority of DeFi applications are open source to gain users' trust, some remain closed source, making our Type-I prompt inapplicable. To address this, we developed a Type-II prompt template to infer the trend of price changes in closed source two-token liquidity pools. Our observation is that the majority of two-token liquidity pools use the CPMM as their underlying price model. Therefore, the primary distinction between the Type-I prompt and the Type-II prompt lies in replacing the Instruction 1 with a description of the liquidity pool, informing the LLM that the pool's price model aligns with CPMM. It is worth noting that the liquidity pool is automatically identified during transaction analysis, which will be introduced in §V. To evaluate its effectiveness, we applied Type-II on a test dataset [29] of 100 price changes on CPMM, and it reached 97% accuracy, close to the 99% accuracy of Type-I. This demonstrates that, if applied to previously identified two-token liquidity pools, which commonly adopt CPMM, Type-II is substantially effective comparable to Type-I, while also adapting to closed-source pools. More details about the Type-II prompt and a case study of using it are available in [28].

V. DeFi Operations

The standalone fluctuations in token prices are meaningless; they need to be considered within the DeFi context to serve as evidence for detecting price manipulation. However, the raw transactions obtained from the blockchain consist solely of low-level information, such as token transfer actions and smart contract invocations. There exists a gap between raw transactions and high-level DeFi semantics.

To bridge this gap, we first model token transfer actions using a directed graph (§V-A), and then recover high-level DeFi operations from it (§V-B). Since our detection is based on a single transaction, it should be noted that all described operations are derived from one raw transaction, and we do not consider the DeFi operations expressed by the combination of multiple raw transactions. Based on our study of the top-10 high-value DeFi applications (the full list could be found in [28]) across three categories — Decentralized EXchange (DEX), Lending, Yield Farming, with active transactions in each category, due to the susceptibility to front-running across

multiple transactions and the atomicity of transactions ensuring complete execution of operations, only a very few DeFi operations span multiple transactions.

A. Transfer Graph Construction

We define the Transfer Graph (TG) (Definition 2), a directed graph where the edges represent transfer actions (Definition 1) and the vertices represent related accounts, to model transfer actions within each user invocation.

Definition 1 (Transfer): A transfer $T := \langle s, r, t, v \rangle$, if performed successfully, deducts amount $v \in \mathbb{N}$ of token $t \in Addr$ from the sender's account $s \in Addr$ and the balance of token t in the receiver's account $r \in Addr$ increases by v.

Definition 2 (Transfer Graph): A *Transfer Graph* (*TG*) is a tuple ($\mathcal{A}$,$\mathcal{E}$), where $\mathcal{A}$ is the set of all accounts (including EOAs, CAs and $\emptyset$) involved in a user invocation, $\mathcal{E}$ is the set of directed edges, i.e., $\mathcal{E} = \{E_1, ..., E_m\} \subseteq \mathcal{A} \times \mathcal{A}$, where each $E_i := \langle j, T_k \rangle$, j is the time index of T_k, $T_k \in \mathcal{T}$, $\mathcal{T}$ is the set of all transfer actions involved in the user invocation, i.e., $\mathcal{T} = \{T_1, ..., T_n\}$, where $T_i.s, T_i.r \in \mathcal{A}$ for each T_i.

According to our categorization, a transfer action can be one of three types: *transferring*, *burning*, and *minting token*. All transfer actions can be expressed as "*Sender* transfers *amount* of *token* to *Receiver*." In the *transferring token*, all accounts involved must be either EOAs or CAs, and must not be a zero address or a dead address (we uniformly denote these two special addresses by $\emptyset$). Meanwhile, the *receiver* in *burning token* and the *sender* in *minting token* actions must be $\emptyset$.

Fig. 4, as demonstrated in step ①, illustrates the construction of a TG from the raw transaction of a user invocation. This user invocation includes six contract accounts and a collection of user-controlled accounts UC, which includes EOAs and CAs, along with seven *transferring token* actions. The *Sender* and *Receiver* of a transfer action are connected by a directed edge, from the *Sender* to the *Receiver*, with a time index to indicate the order of occurrence. T_1, from one of the user-controlled accounts in UC to CA_1, is the first transfer action in this user invocation. Before CA_1 transfers tokens to CA_2 through T_4, user-controlled accounts initiate two transfers to CA_4 and CA_5 respectively, resulting in T_4 having a larger time index compared to T_2 and T_3. Similarly, since T_6 occurs between T_5 (with a time index of 5) and T_7 (with a time index of 7), its time index is set to 6.

Compared to the CFT approach used in DeFiRanger [4], which models invocation and transfer actions within raw transactions, our TG provides finer granularity. TG represents transfers as a directed graph while preserving their temporal order, enabling it to capture more complex operations more effectively. In our evaluation on 1,000 real-world Ethereum transactions, TG achieved a TPR ($\frac{\#TP}{\#TP+\#FN}$) of 0.912, significantly higher than CFT's 0.559, while maintaining comparable precision ($\frac{\#TP}{\#TP+\#FP}$), i.e., 0.984 vs. 0.991. TG further demonstrated substantial advantages in recovering complex swap operations involving multiple relayers, achieving an 89.3% improvement over CFT. More details about the experimental setup and analysis are available in [28]. Compared

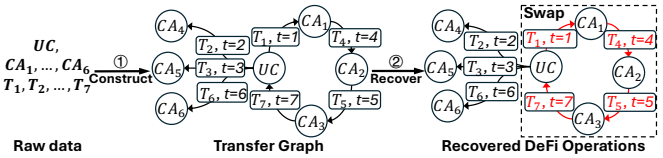


Fig. 4. An illustrative example showing the workflow of recovering DeFi operations. UC: a collection of user-controlled accounts, including EOAs and CAs; CA_i: the i-th contract account; T_i: the i-th *transferring token* action.

to prior fund-flow graph approaches, such as [30], TG differs in two key aspects: (i) it is constructed at the level of user invocations rather than transactions, which may contain multiple invocations, and (ii) it is designed for high-level DeFi operation recovery, whereas those primarily target behavior analysis (e.g., arbitrage).

B. DeFi Operation Inference

Based on our in-depth analysis of the top-30 DeFi apps, we identify six DeFi operations that need to be recovered: 1) *Swap*, primarily from DEXs such as Uniswap; 2) *Deposit*, defined by yield-farming and lending apps such as AAVE and Pendle; 3) *Withdraw*, mainly from yield-farming apps; 4) *Borrow*, based on lending protocols like Compound; 5) *Stake* and 6) *Claim*, both predominantly from yield-farming apps like Convex Finance.

Swap involves a user exchanging one token ($Token_{in}$) for another ($Token_{out}$) from a liquidity pool. Relevant contract accounts, excluding user-controlled ones, are noted as liquidity pools for price trend analysis when source code is unavailable. In a **Deposit**, a user transfers a token ($Token_{deposit}$) to a yield-farming or lending protocol and receives a proof token ($Token_{proof}$) through minting. **Withdraw** occurs when a user retrieves tokens ($Token_{withdraw}$) from a protocol by burning a proof token ($Token_{proof}$). **Borrow** refers to a DeFi operation where a borrower receives a token ($Token_{borrow}$) and incurs a debt evidenced by another token ($Token_{debt}$), issued through minting. **Stake** happens in yield-farming protocols like Convex Finance [35], where users deposit tokens and can earn rewards without receiving a minted proof token. **Claim** enables users to retrieve staked tokens ($Token_{claim}$) and bonuses without burning any tokens, unlike the *Withdraw* operation which requires burning tokens to retrieve assets.

An Operation Recovering Example. To recover DeFi operations from the TG, we design and employ a search algorithm based on directed graphs for each operation. Fig. 4 illustrates that, to recover a *Swap* operation, we use the depth-first search algorithm to identify cycles that both start and end at user-controlled accounts. These cycles must satisfy three constraints: (i) the transfer action of each edge must be *transferring token*; (ii) the time index of each edge must be monotonically increasing; and (iii) the token involved in the first transfer action must differ from the token in the last transfer action. Therefore, from the well-constructed TG, we can recover the *Swap* operation expressed as $UC \rightarrow T_1 \rightarrow CA_1 \rightarrow T_4 \rightarrow CA_2 \rightarrow T_5 \rightarrow CA_3 \rightarrow T_7 \rightarrow UC$, and we label CA_1, CA_2, and CA_3 as liquidity pools.

TABLE I
PRICE CHANGE INFORMATION-DIRECTED ATTACK PATTERNS.

Type	Pattern	Details
Buy & Sell [31]	I	1) Swap $Token_x$ to $Token_y$ through $Pool_{buy}$ 2) The price of $Token_y$ in $Pool_{sell}$ increases / The price of $Token_z$ in $Pool_{sell}$ decreases 3) Swap $Token_y$ to $Token_z$ through $Pool_{sell}$
	II	1) The price of $Token_x$ in $Pool_{buy}$ increases / The price of $Token_y$ in $Pool_{buy}$ decreases/ The price of $Token_y$ in $Pool_{sell}$ increases / The price of $Token_z$ in $Pool_{sell}$ decreases 2) Swap $Token_x$ to $Token_y$ through $Pool_{buy}$ 3) Swap $Token_y$ to $Token_z$ through $Pool_{sell}$
Deposit & Borrow [32]	III	1) Deposit $Token_x$ into $Contract_{deposit}$ and get $Token_y$ as credential 2) The price of $Token_x$ in $Contract_{borrow}$ increases / The price of $Token_z$ in $Contract_{borrow}$ decreases 3) Borrow $Token_z$ using $Token_x$ as collateral from $Contract_{borrow}$
	IV	1) The price of $Token_x$ in $Contract_{borrow}$ increases / The price of $Token_z$ in $Contract_{borrow}$ decreases 2) Deposit $Token_x$ into $Contract_{deposit}$ and get $Token_y$ as credential 3) Borrow $Token_z$ using $Token_x$ as collateral from $Contract_{borrow}$
Stake & Claim [33]	V	1) Stake $Token_x$ into $Contract_{stake}$ 2) The price of $Token_y$ in $Contract_{claim}$ decreases 3) Claim $Token_y$ from $Contract_{claim}$
	VI	1) The price of $Token_x$ in $Contract_{stake}$ increases / The price of $Token_y$ in $Contract_{claim}$ decreases 2) Stake $Token_x$ into $Contract_{stake}$ 3) Claim $Token_y$ from $Contract_{claim}$
Deposit & Withdraw [34]	VII	1) Deposit $Token_x$ into $Contract_{deposit}$ and get $Token_y$ as credential 2) The price of $Token_y$ in $Contract_{withdrawal}$ increases / The price of $Token_z$ in $Contract_{withdrawal}$ decreases 3) Withdraw $Token_z$ from $Contract_{withdrawal}$ by burning $Token_y$
	VIII	1) The price of $Token_x$ in $Contract_{deposit}$ increases / The price of $Token_y$ in $Contract_{deposit}$ decreases / The price of $Token_y$ in $Contract_{withdrawal}$ increases / The price of $Token_z$ in $Contract_{withdrawal}$ decreases 2) Deposit $Token_x$ and get $Token_y$ as credential 3) Withdraw $Token_z$ from $Contract_{withdrawal}$ by burning $Token_y$

VI. PRICE MANIPULATION DETECTION

The price change information and high-level DeFi operations recovered from §IV and §V, respectively, are finally checked against the detection rules (c.f. Table I). We analyze four prevalent types of DeFi protocols and their associated attack instances. Specifically, we examine ElephantMoney [31] for *DEX*, Cream Finance [32] for *lending protocols*, ATK [33] for *staking-based yield farming protocols*, and Harvest [34] for *deposit-based yield farming protocols*, which collectively incurred a loss of $163.3M.

Based on this in-depth analysis, we identify four attack types targeting different DeFi protocols and their eight generalized attack patterns, as depicted in Table I. These attack types are *Buy & Sell*, *Deposit & Borrow*, *Stake & Claim*, and *Deposit & Withdraw*, with each type corresponding to two specific attack patterns. More details about these attack types and patterns are available in our supplementary material [28].

We implement DeFiScope with about 3,900 lines of Python code. DeFiScope currently supports two blockchains, i.e., Ethereum [36] and BSC [37], which account for over 60% of total value locked (TVL) among all the blockchains [38]. To obtain raw data from specific transactions, we utilize blockchain node APIs facilitated by QuickNode [39], an external RPC service provider.

VII. EVALUATION

We aims to evaluate the following research questions (RQs):

- **RQ1**: How effectively does DeFiScope detect price manipulation attacks compared with the existing state-of-the-arts?
- **RQ2**: How significantly does the fine-tuning technique promote the accuracy of DeFiScope?
- **RQ3**: How practically and efficiently does DeFiScope detect price manipulation attacks in a real-world setting?

Datasets. To address these RQs, we collect three datasets from real DeFi transactions on Ethereum and BSC to evaluate DeFiScope, as shown in Table II. Specifically, we use the first dataset *D1*, which comprises 95 transactions of real-world price manipulation attacks from 90 DeFi applications, to evaluate both RQ1 and RQ2. To demonstrate DeFiScope's practicality for RQ3, we use the second dataset *D2*, consisting of 968 suspicious transactions collected by our industry partner. Furthermore, to measure DeFiScope's time overhead when deployed in a realistic setting with both suspicious and benign transactions, we mix 968 suspicious transactions from *D2* with 96,800 benign transactions in the third dataset *D3*.

For *D1*, we scraped data from multiple sources. Initially, we included all 54 price manipulation transactions identified by DeFort [5]. We also collected all 55 price manipulation attacks documented by the renowned DeFi security GitHub repository, DeFiHackLabs [40], from October 26, 2020, to October 11, 2023. To further expand our dataset, we acquired 31 transactions confirmed as price manipulation attacks from our industry partner. After removing duplicates among the three sources of data, we ultimately obtained 95 price manipulation attacks for *D1*, which collectively caused $381.16M in losses on Ethereum and BSC. We also measure the distribution and monetary losses of the 95 attacks in *D1* across our eight attack patterns described in §VI, as shown in Table III.

D2 comprises 968 suspicious transactions provided by our industry partner, a Web3 security company, which monitors blockchain transactions in real-time and automatically flags transactions that yield significant profits for initiators. All these transactions are sufficiently complex and potentially

TABLE II
THE BENCHMARK DATASETS USED FOR EVALUATION.

Dataset	RQs
D1: 95 historical real-world attacks	RQ1, RQ2
D2: 968 suspicious transactions	RQ3
D3: 96,800 benign transactions	RQ3

TABLE III
SUMMARY OF DIFFERENT ATTACK PATTERNS IN *D1*.

Pattern	I	II	III	IV	V	VI	VII	VIII
#Case	49	20	5	6	1	2	11	1
Loss($)	55.3M	70M	141.2M	43M	61K	83K	71.4M	9K

involve price manipulation. While manually confirming these transactions is labor-intensive and prone to errors, 155 of them have been confirmed to belong to other vulnerabilities, such as Reentrancy (OpenLeverage [41]), Unverified User Input (YIELD [42]), and Access Control Bugs (SafeMoon [43]), by developers and industry partners. In this way, we can stress-test DeFiScope's detection capabilities when mixed with other types of attack transactions.

Furthermore, to assess DeFiScope's false alarms on benign transactions and to measure its time overhead in a realistic setting, we need to collect a large number of benign transactions, as the most majority of real-world transactions are still benign. In the absence of empirical data on the ratio of suspicious to benign transactions, we adopt a conservative yet reasonable ratio of 1:100, assuming one suspicious transaction for every 100 benign transactions. Thus, with 968 suspicious transactions in *D2*, we require 96,800 benign transactions for *D3*. To this end, we randomly sampled these 96,800 transactions from DeFort [5]'s dataset of 428,523 benign transactions, which includes 384,143 benign transactions on Ethereum and BSC. At this sample size, we achieve a 99.999% confidence level with a margin of error of 0.625%.

Minimized Data Leakage. DeFiScope utilizes LLMs to assess token price changes solely by using price calculation functions and token balance changes, without any knowledge of attack transactions. Even if related attacks appeared in pre-training, evaluating price change tendencies in this manner falls outside its training data, resulting in no or minimized data leakage.

Experimental Setup. All experiments were conducted on a desktop computer running Ubuntu 20.04, powered by an Intel® Xeon® W-2235 CPU (3.80 GHz, 6 cores, and 12 threads) and equipped with 16 GB of memory. For LLMs, we *by default* use OpenAI's GPT family models but also explore open-source SFT model tuning in §VIII, as explained in §IV-A. DeFiScope by default uses GPT-3.5-Turbo (`GPT-3.5-turbo-1106`) for its well-recognized cost-performance balance, but we also conduct an ablation study in §VII-B to test the more advanced GPT-4o (`GPT-4o-2024-08-06`) for fine-tuning. For the LLM configuration, DeFiScope employs nucleus sampling [44] with a `top-p` value of 1 and sets the `temperature` to 0, ensuring highly deterministic responses for each prompt, although each was run only once.

A. RQ1: Detection Effectiveness

To answer RQ1, we evaluate DeFiScope and compare it with three SOTA tools, DeFiTainter [7], DeFort [5], and DeFiRanger [4], specialized in detecting price manipulations using the dataset *D1*. Because DeFiRanger is not open-source, we use the results reported in their paper for the attacks they evaluated; for other attacks, we re-implemented their approach based on the description [4] and conducted our evaluation. Although DeFort is also not open-source, we obtained a copy of code from its authors to run experiments. FlashSyn [45] and SMARTCAT [46] were excluded as baselines because FlashSyn targets broader attacks beyond price manipulations and lacks guidelines for detecting new attacks, while SMARTCAT was unpublished and not open-source at the time of evaluation.

Table IV presents the detection results. The first four columns list the name of the victim protocol, chain deployed, hack date, and resulting loss, respectively. *DS*, *DT*, *DF* and *DR* denote DeFiScope, DeFiTainter, DeFort and DeFiRanger, respectively. Note that one protocol may face multiple attacks, so we add numerical suffixes to protocol names to differentiate them. We use ✓ to indicate an attack can be successfully detected by a tool, and ✗ to indicate a detection failure.

DeFiScope can detect most of the price manipulation attacks. It achieves a recall rate of 80%, outperforming all other tools. Overall, DeFiScope detected 76 attacks, followed by DeFort with 50 and DeFiRanger with 49 attacks, respectively, while DeFiTainter detected only 34 attacks. In total, DeFiScope failed to detect 19 cases. We manually investigated their root cause and found: (a) 11 cases should be detectable but were missed by DeFiScope due to the limitation of LLM reasoning, non-ERC20 token and the restriction to only single-transaction analysis, and (b) 8 cases cannot be detected because they do not satisfy the source code requirement of DeFiScope due to either ① missing source code or ② compilation errors, as marked in Table IV. Moreover, the statistical significance of DeFiScope's performance improvements over each baseline is confirmed by the McNemar Test [47], with all the resulting p-values being below 0.05 (see details in [28]). Fig. 5 details the performance of each tool on the evaluated DeFi protocols across four application categories using different price models. Compared to other tools, DeFiScope performs the best in every application category. Particularly, it achieves the highest recall, 90.7%, in the Token category. However, DeFiScope yields a low recall rate in Lending-related protocols, though still higher than all existing tools. Through further analysis, we identifies that 3 out of 12 attacks (InverseFinance_2, SanshuInu, and VesperFinance) are cross-transaction attacks that DeFiScope is unable to detect. Additionally, one attack (TIFIToken) involves exploiting a closed-source custom price model, and the InverseFinance_1 attack, a false negative, will be detailed below.

After analyzing 11 detectable but missed attacks, we discovered that 8 out of 11 are cross-transaction attacks [33], [48], [49], [50], [51], [52], [53], [54], where detection was unsuccessful because DeFiScope is based on analyzing indi-

TABLE IV
DETECTION RESULTS FOR 95 GROUND-TRUTH DEFI PRICE MANIPULATION ATTACKS.

Protocol	Chain	Date	Loss	DS	DT	DF	DR
AES	BSC	07-Dec-22	60K	✓	✗	✓	✗
APC	BSC	01-Dec-22	6K	✓	✗	✓	✗
APEDAO	BSC	18-Jul-23	7K	✓	✓	✗	✓
ApeRocket	BSC	14-Jul-21	1.26M	①	✓	✓	✗
ARK	BSC	23-Mar-24	201K	✗	✗	✗	✗
ArrayFinance	ETH	18-Jul-21	516K	②	✗	✗	✓
ATK	BSC	12-Oct-22	61K	✗	✗	✓	✗
AutoSharkFinance_1	BSC	29-Oct-21	2M	✓	✓	✓	✓
AutoSharkFinance_2	BSC	24-May-21	750K	✓	✓	✓	✗
BabyDoge	BSC	28-May-23	137K	✓	✗	✗	✗
Bamboo	BSC	04-Jul-23	117K	✓	✗	✗	✓
BBOX	BSC	16-Nov-22	12K	✓	✗	✓	✗
BDEX	BSC	30-Oct-22	3K	②	✓	✓	✗
bDollar	BSC	21-May-22	730K	✓	✗	✗	✓
BEARNDAO	BSC	05-Dec-23	769K	✓	✗	✓	✓
BeltFinance_1	BSC	29-May-21	408K	✓	✓	✗	✓
BeltFinance_2	BSC	29-May-21	6.23M	✓	✓	✗	✓
BFCToken	BSC	09-Sep-23	38K	✓	✗	✗	✓
BGLD	BSC	12-Dec-22	18K	✓	✓	✓	✗
BH	BSC	11-Oct-23	1.27M	✓	✓	✗	✓
BTC20	ETH	19-Aug-23	47K	✓	✗	✗	✗
BXH	BSC	28-Sep-22	40K	✓	✓	✓	✗
bZx	ETH	18-Feb-20	350K	✓	✗	✓	✓
Carson	BSC	26-Jul-23	150K	①	✗	✓	✓
Cellframe	BSC	01-Jun-23	76K	✓	✓	✗	✓
CheeseBank	ETH	06-Nov-20	3.3M	✓	✓	✓	✓
ConicFinance	ETH	21-Jul-23	3.25M	✓	✗	✗	✓
CreamFinance	ETH	27-Oct-21	130M	✓	✓	✓	✗
CS	BSC	23-May-23	714K	✓	✓	✗	✓
Cupid	BSC	31-Aug-22	78K	✓	✓	✓	✓
DFS	BSC	30-Dec-22	2K	✓	✗	✓	✓
Discover	BSC	06-Jun-22	11K	✓	✗	✗	✗
DotFinance	BSC	25-Aug-21	430K	✓	✓	✓	✗
EAC	BSC	29-Aug-23	17K	✓	✗	✓	✓
EGDFinance	BSC	07-Aug-22	36K	✓	✓	✓	✗
ElephantMoney	BSC	12-Apr-22	11.2M	✓	✗	✓	✗
Eminence	ETH	29-Sep-20	7M	✓	✓	✓	✗
ERC20TokenBank	ETH	31-May-23	111K	✓	✗	✗	✓
FFIST	BSC	19-Jul-23	91K	✓	✗	✓	✗
GDS	BSC	03-Jan-23	180K	✓	✓	✓	✗
GPT	BSC	24-May-23	42K	✓	✗	✓	✗
Groker20	ETH	10-Nov-23	68K	✓	✗	✗	✓
GymDeFi	BSC	09-Apr-22	312K	✓	✗	✓	✓
Hackerdao	BSC	24-May-22	65K	✓	✗	✓	✓
Harvest	ETH	26-Oct-20	21.5M	②	✓	✓	✓
IndexedFinance	ETH	14-Oct-21	16M	✗	✗	✗	✓
INUKO	BSC	14-Oct-22	50K	✗	✗	✓	✗
InverseFinance_1	ETH	16-Jun-22	1.26M	✗	✗	✗	✓
InverseFinance_2	ETH	02-Apr-22	15.6M	✗	✗	✗	✗
LaunchZone	BSC	27-Feb-23	320K	✓	✗	✓	✗
LUSD	BSC	07-Jul-23	9K	✓	✗	✓	✗
LW_1	BSC	12-May-23	50K	✓	✗	✗	✓
LW_2	BSC	12-May-23	48K	✓	✗	✗	✓
Mars	BSC	16-Apr-24	100K	✓	✗	✗	✗
MBC	BSC	29-Nov-22	6K	✓	✓	✓	✗
MerlinLab	BSC	29-Jun-21	628K	✓	✗	✗	✗
MonoXFinance	ETH	30-Nov-21	31M	✓	✗	✗	✓
MRGtoken	ETH	08-Nov-23	12K	✓	✗	✗	✓
NeverFall	BSC	02-May-23	74K	✓	✓	✗	✗
Nmbplatform	BSC	14-Dec-22	76K	②	✓	✓	✓
NOVO_1	BSC	29-May-22	76K	✓	✗	✗	✓
NOVO_2	BSC	29-May-22	65K	✓	✗	✓	✓
PancakeBunny	BSC	19-May-21	45M	✓	✓	✓	✗
PancakeHunny	BSC	20-Oct-21	1.93M	②	✗	✗	✗
PLPManager	BSC	24-Jul-23	900K	✓	✓	✗	✗
PLTD	BSC	17-Oct-22	24K	✓	✗	✓	✓
RoeFinance	ETH	11-Jan-23	80K	✓	✗	✗	✗
SanshuInu	ETH	20-Jul-21	111K	✗	✓	✓	✗
SATX	BSC	16-Apr-24	29K	✓	✗	✗	✓
SellToken	BSC	11-Jun-23	100K	✓	✗	✗	✓
SpaceGodzilla	BSC	13-Jul-22	25K	✓	✗	✓	✓
SpartanProtocol	BSC	01-May-21	30M	✓	✗	✗	✓
Starlink	BSC	16-Feb-23	12K	✓	✗	✓	✓
StarWallets	BSC	17-Apr-24	33K	✗	✗	✗	✗
STM	BSC	06-Jun-24	14K	✓	✓	✗	✗
SturdyFinance	ETH	12-Jun-23	800K	✓	✗	✗	✗
SVT	BSC	26-Aug-23	400K	✓	✓	✗	✓
SwapX	BSC	27-Feb-23	1M	✗	✗	✓	✗
TIFIToken	BSC	10-Dec-22	51K	①	✓	✓	✗
UN	BSC	06-Jun-23	26K	✓	✗	✗	✓
UPSToken	ETH	18-Jan-23	45K	✓	✓	✓	✓
Upswing	ETH	17-Jan-23	36K	✓	✓	✓	✗
uwerx_network	ETH	02-Aug-23	324K	✓	✗	✗	✓
UwULend	ETH	10-Jun-24	19M	✓	✗	✗	✗
ValueDeFi	ETH	14-Nov-20	6M	✓	✓	✓	✓
VesperFinance	ETH	02-Nov-21	2M	✗	✗	✗	✗
VINU	ETH	06-Jun-23	6K	✓	✗	✓	✗
WarpFinance	ETH	17-Dec-20	7.8M	✓	✓	✗	✓
WGPT	BSC	12-Jul-23	80K	✓	✗	✓	✓
WienerDoge	BSC	25-Apr-22	30K	✓	✓	✓	✓
XSTABLE	ETH	09-Aug-22	56K	✓	✗	✓	✗
Z123	BSC	22-Apr-24	136K	✓	✗	✗	✓
Zoompro	BSC	05-Sep-22	61K	✗	✗	✓	✗
ZS	BSC	08-Oct-23	14K	✓	✗	✓	✓
Zunami	ETH	13-Aug-23	2M	✓	✗	✗	✗

vidual transactions. For Zoompro, detection failure occurred because the token involved in the transaction did not adhere to the ERC20 token standard, resulting in the transfer event not being identified. The remaining two attacks, i.e., "IndexedFinance" and "InverseFinance_1" cases, were subject to in-depth analysis. The detection incapability for IndexedFinance was due to its use of an extremely complicated pricing mechanism that involves exponential calculations in its price-related function `joinswapExternAmountIn`. In the case of InverseFinance, the attacker exploited the flawed price dependency when calculating the price of tokens deposited as collateral, where the collateral price is based on the balance of multiple tokens in the liquidity pool and off-chain price oracles. As both cases require precise quantitative calculations, DeFiScope is limited by the current LLMs' constrained capacity for scientific computation. A potential solution could be to integrate with Program-Aided Language models (PAL) [55], guiding the LLM to generate scripts for necessary calculations and executing them to obtain the result.

Among the 8 undetectable transactions, 3 were not analyzed successfully due to the unavailability of the code of price calculation functions since our method relies on code-level analysis; five cases involved compilation errors during the extraction of price calculation functions using Slither [56], an off-the-shelf static analyzer for Solidity and Vyper, thereby prematurely terminating the detection process. These limitations are not inherent to DeFiScope's methodology and can be mitigated by automated or semi-automated techniques, e.g., code decompilation [57], [58], [59] and manual intervention.

B. RQ2: Ablation Study

In this RQ, we investigate how fine-tuning can enhance DeFiScope's detection accuracy on the same *D1* dataset, as well as the impact and cost of fine-tuning different GPT

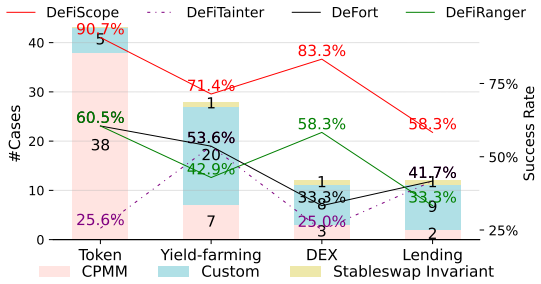
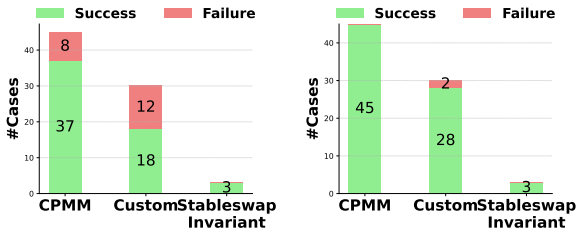
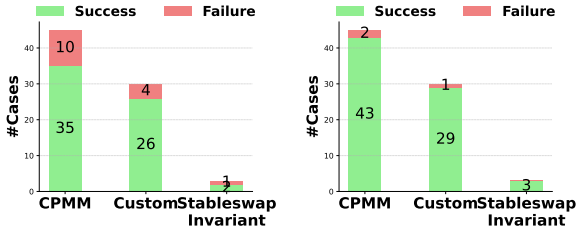


Fig. 5. The categorized detection results on DeFi protocols.



(a) GPT-3.5-Turbo w/o fine-tuning (b) GPT-3.5-Turbo w/ fine-tuning



(c) GPT-4o w/o fine-tuning (d) GPT-4o w/ fine-tuning

Fig. 6. Effectiveness of fine-tuning for 78 attacks that go through DeFiScope's LLM inference. Cases not plotted in the figures include eight due to missing source code and compilation errors, and nine due to cross-transaction issues and non-adherence to the ERC20 token standard, as mentioned in §VII-A.

models. To this end, we test four settings shown in Fig. 6: (a) the original GPT-3.5-Turbo without fine-tuning, (b) GPT-3.5-Turbo with fine-tuning, used in RQ1, (c) the original GPT-4o without fine-tuning, and (d) GPT-4o with fine-tuning.

Fig. 6 shows that fine-tuning significantly enhances DeFiScope's ability to detect attacks, with the fine-tuned versions of GPT-3.5-Turbo and GPT-4o detecting 18 (31%) and 12 (19%) more attacks, respectively. It also indicates that fine-tuning provides a more noticeable improvement for less powerful models, such as GPT-3.5-Turbo, compared to stronger models like GPT-4o. With fine-tuning, the detection success rate for attacks targeting CPMM increases to 100% with GPT-3.5-Turbo and 95.6% with GPT-4o[2]. This very high rate can be attributed to the use of CPMM during the fine-tuning training phase, enabling the model to effectively handle it.

Although DeFiScope's fine-tuned models were trained using only CPMM data, we find that they also exhibit strong transfer learning capabilities for attacks targeting custom price models. Specifically, the detection success rate for attacks targeting custom price models increases from 60% to 93.3% (a relative increase of 55.5%) for GPT-3.5-Turbo and from 86.7% to 96.7% (a relative increase of 11.5%) for GPT-4o. Further analysis of the LLM responses generated during detection reveals that the most significant difference introduced by fine-tuning is that the fine-tuned model strictly adheres to the CoT approach we specified. This involves initially extracting the price model from the given code and then conducting inference based on the extracted model along with the provided information about balance changes. In contrast, although the original model produces the final evaluation scores, it does not strictly follow the instructions of the CoT prompts.

[2]GPT-4o does not exhibit an advantage with CPMM but shows a clear advantage with custom price models, especially for the original GPT-4o.

TABLE V
COMPARISON BETWEEN GPT-3.5-TURBO AND GPT-4O.

Model	*TP*	*Recall*	Ave. Cost($)/ per inference	Fine-tuning Cost($)
GPT-3.5 w/o fine-tuning	58	0.61	0.0023	-
GPT-3.5 w/ fine-tuning	76	0.80	0.0107	8
GPT-4o w/o fine-tuning	63	0.66	0.008	-
GPT-4o w/ fine-tuning	75	0.79	0.0131	25

LLM Costs. Table V highlights that while the original GPT-4o generally performs better than GPT-3.5-Turbo, this performance gap can be largely minimized through fine-tuning. However, regarding costs, GPT-3.5-Turbo has a clear advantage, which is why DeFiScope uses GPT-3.5-Turbo for fine-tuning by default instead of GPT-4o. Specifically, the fine-tuning cost of GPT-3.5-Turbo is 68% lower than that of GPT-4o, and the per-request inference cost of GPT-3.5-Turbo is merely $0.0107 [60], which is also 18% cheaper than that of GPT-4o. Therefore, for a tradeoff between cost and performance, we recommend using GPT-3.5-Turbo.

C. RQ3: Real-World Practicality

To be practical in a real-world setting, DeFiScope not only needs to maintain high detection rates for true attacks but also minimize false alarms for benign transactions. In this RQ, we evaluate this aspect of DeFiScope and the associated time overhead using the datasets *D2* and *D3*, which were introduced in the prologue of §VII. Specifically, *D2* comprises 968 suspicious transactions with various attacks (i.e., not limited to price manipulations), and *D3* includes 96,800 benign transactions collected from DeFort [5]'s dataset.

For *D2*, DeFiScope flagged 153 price manipulation attacks out of 968 suspicious transactions. To robustly confirm these potential attacks, we cross-referenced them with attack reports or alerts published by security companies through their official channels [61], [62], [63], [64], thereby verifying the root causes of the attacks. Using this method, we confirm that 66 of them are previously reported price manipulation attacks. For the remaining cases, we conducted comprehensive and in-depth analysis by combining manual review with ancillary evidence, such as identifying whether the EOA initiating the transaction was marked as a hacker by blockchain explorers [65], [66], [67]. For newly discovered incidents, we manually verified: (i) whether abnormal token price or exchange rate changes occured, and (ii) whether the user gained profit from such changes. Cases satisfying both criteria were labeled as price manipulation attacks. Finally, we discovered 81 previously unknown historical incidents, which were

TABLE VI
COMPARISON BETWEEN GPT TUNING AND PHI-3'S SFT.

Model	*Recall*	Model	*Recall*
GPT-3.5 w/o fine-tuning	0.61	GPT-3.5 w/ fine-tuning	0.80
GPT-4o w/o fine-tuning	0.66	GPT-4o w/ fine-tuning	0.79
Phi-3 w/o fine-tuning	0.52	Phi-3 w/ fine-tuning [70]	0.66

neither reported by security companies nor tagged as malicious transactions by blockchain explorers, and identified a total of six false positive cases (the full list can be found in [28]). Of these, five are benign transactions initiated by the same EOA and exhibiting identical invocation flows, while the last one is indeed an attack due to a logic issue rather than a price manipulation attack, resulting in a precision of 96% (147/153) on suspicious transactions. In comparison, DeFort identified only 58 attacks, of which 9 were false positives labeled as other vulnerabilities by security researchers. Moreover, DeFort failed to detect 114 of the malicious transactions flagged by DeFiScope, including 54 that are officially confirmed attacks.

We further analyze DeFiScope's false alarms in a realistic setting mixing 968 suspicious transactions in *D2* with 96,800 benign transactions in *D3* (see the earlier dataset setup in the prologue of this section). For each transaction, we set the maximum scan time to 300 seconds. The results reveal that DeFiScope achieves zero false alarms on benign transactions in *D3*, and 6 false positives on *D2* (as mentioned above), with an average of 2.5 seconds per transaction across all transactions. More than half the time is spent on price change inference that uses up an average of 1.40 seconds, while static analysis takes slightly less, around 0.97 seconds on average. The other steps within DeFiScope incur negligible cost. A detailed time cost breakdown is available in [28]. This highlights DeFiScope's potential for large-scale, daily on-chain monitoring scenarios.

VIII. DISCUSSION

Cross-LLM Generalizability. To show DeFiScope's methodology is not limited to OpenAI's fine-tuning paradigm [22], we performed SFT fine-tuning [23] on a open-source Phi-3 [68] model using LoRA [24], on a sever with 4 Nvidia H800 GPUs. We used *all* 1,000 price change samples mentioned in §IV, split 8:1:1 into training, evaluation, and test sets. As shown in Table VI, LoRA-based fine-tuning enabled Phi-3 to outperform GPT-3.5-Turbo (without fine-tuning) and achieve performance comparable to GPT-4o (without fine-tuning). Full hyperparameter settings and results are available in our repository [69].

Cross-Blockchain Generalizability. Although our evaluation primarily focused on Ethereum and BSC, which together account for the majority of observed price manipulation attacks, DeFiScope is generalizable to EVM-compatible blockchains, such as Polygon [71] and Arbitrum [72]. DeFiScope successfully detected seven price manipulation attacks on these chains. Full details of these cases are available in [28].

Closed-source contracts and cross transaction attacks. The availability of price model code can affect DeFiScope's detection accuracy. According to our study, most DeFi applications are open-source to gain user trust. To mitigate issues with closed-source price models, we design the Type-II prompt to cover price models in those closed-source liquidity pools. Currently, DeFiScope detects only single-transaction attacks, while some attacks span multiple transactions to bypass protocol time restrictions [73]. Detecting such cases is challenging. For instance, attacks like [48] span over 48 hours (about 57,000 blocks), making it difficult to trace all related transactions. Fortunately, our study observed that such attacks are far less common than single-transaction ones.

IX. RELATED WORK

On-chain Security Analysis. Prior efforts [74], [75], [76], [77], [78] have studied blockchain threats and DeFi attacks. Some transaction-based systems are proposed to mine vulnerable transaction sequences [79], [80], explore arbitrage opportunities [81], detect malicious phishing [82], and simulate attacks to prevent intrusions [83]. Tools like FlashSyn [45], DeFiRanger [4], DeFiTainter [7], and DeFort [5] are capable of detecting attacks associated with price manipulation. FlashSyn uses numerical approximation to synthesize malicious contracts that target DeFi apps through price manipulation attacks.

LLMs for Smart Contract Security. LLMs have become powerful tools in blockchain security. GPTScan [84] leverages LLMs with static analysis to detect logical vulnerabilities. BlockGPT [85] uses LLMs to rank transaction anomalies for real-time intrusion detection. LLM4Vuln [86] improves LLM reasoning for vulnerability analysis. iAudit [25] combines fine-tuning with agents for intuitive auditing. PropertyGPT [87] uses LLMs for retrieval-augmented property generation. To our knowledge, DeFiScope is the first tool that uses LLMs specifically designed for detecting price manipulation attacks.

X. CONCLUSION

In this paper, we introduced DeFiScope, the first tool that utilizes LLMs specifically for detecting price manipulation attacks. DeFiScope employs LLMs to intelligently infer the trend of token price changes based on balance information within transaction executions. To strengthen LLMs in this aspect, we simulated on-chain transaction data and fine-tuned a DeFi price-specific LLM. We also proposed a graph-based method to recover high-level DeFi operations and systematically mined eight price manipulation patterns. Our evaluation demonstrated DeFiScope's superior performance over SOTA approaches and real-world impact. Future work includes better handling of closed-source price calculation functions.

ACKNOWLEDGMENT

We thank all reviewers for their constructive feedback. This research was supported by Lingnan Grant SUG-002/2526, HKUST TLIP Grant FF612, the National Natural Science Foundation of China (Project No. 72304232), the Singapore Ministry of Education Academic Research Fund Tier 2 (T2EP20224-0003) and the Nanyang Technological University Centre for Computational Technologies in Finance (NTU-CCTF). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of MOE and NTU-CCTF.

REFERENCES

[1] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu, "Smart contract development: Challenges and opportunities," *IEEE transactions on software engineering*, vol. 47, no. 10, pp. 2084–2106, 2019.

[2] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE transactions on knowledge and data engineering*, vol. 30, no. 7, pp. 1366–1385, 2018.

[3] Cointelegraph, "What is market manipulation in cryptocurrency," https://cointelegraph.com/explained/what-is-market-manipulation-in-cryptocurrency, 2024.

[4] S. Wu, Z. Yu, D. Wang, Y. Zhou, L. Wu, H. Wang, and X. Yuan, "DeFiRanger: Detecting DeFi Price Manipulation Attacks," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[5] M. Xie, M. Hu, Z. Kong, C. Zhang, Y. Feng, H. Wang, Y. Xue, H. Zhang, Y. Liu, and Y. Liu, "Defort: Automatic detection and analysis of price manipulation attacks in defi applications," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 402–414.

[6] Github, "Foundry," https://github.com/foundry-rs/foundry/tree/master, 2024.

[7] Q. Kong, J. Chen, Y. Wang, Z. Jiang, and Z. Zheng, "DefiTainter: Detecting Price Manipulation Vulnerabilities in DeFi Protocols," in *Proc. ACM ISSTA*, 2023.

[8] Uniswap, "Uniswap," https://app.uniswap.org, 2024.

[9] PancakeSwap, "Pancakeswap," https://pancakeswap.finance/swap, 2024.

[10] K. Gogol, C. Killer, M. Schlosser, T. Bocek, B. Stiller, and C. Tessone, "Sok: Decentralized finance (defi)–fundamentals, taxonomy and risks," *arXiv preprint arXiv:2404.11281*, 2024.

[11] Curve.fi, "Stableswap - efficient mechanism for stablecoin liquidity," https://docs.curve.fi/assets/pdf/stableswap-paper.pdf, 2024.

[12] ——, "Curve," https://curve.fi/#/ethereum/swap, 2024.

[13] UwULend, "Uwulend: Decentralized smart lending," https://uwulend.fi, 2024.

[14] Etherscan, "Contract of pool fraxusde," https://etherscan.io/address/0x5dc1bf6f1e983c0b21efb003c105133736fa0743, 2024.

[15] ——, "Contract of pool usdeusdc," https://etherscan.io/address/0x02950460e2b9529d0e00284a5fa2d7bdf3fa4d72, 2024.

[16] ——, "Contract of pool usdedai," https://etherscan.io/address/0xf36a4ba50c603204c3fc6d2da8b78a7b69cbc67d, 2024.

[17] ——, "Contract of pool usdecrvusd," https://etherscan.io/address/0xf55b0f6f2da5ffddb104b58a60f2862745960442, 2024.

[18] ——, "Contract of pool ghousde," https://etherscan.io/address/0x670a72e6d22b0956c0d2573288f82dcc5d6e3a61, 2024.

[19] Curve.fi, "Exponential moving average," https://resources.curve.fi/factory-pools/understanding-oracles/#exponential-moving-average, 2024.

[20] Exvul, "Uwulend attack incident analysis attack brief," https://medium.com/@exvul/uwu-lend-attack-incident-analysisattack-brief-3db51082ec5c, 2024.

[21] Etherscan, "Transaction details of uwulend," https://etherscan.io/tx/0xca1bbf3b320662c89232006f1ec6624b56242850f07e0f1dadbe4f69ba0d6ac3, 2024.

[22] OpenAi, "Openai fine-tuning guideline," https://platform.openai.com/docs/guides/fine-tuning, 2024.

[23] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma *et al.*, "Scaling instruction-finetuned language models," *Journal of Machine Learning Research*, vol. 25, no. 70, pp. 1–53, 2024.

[24] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[25] W. Ma, D. Wu, Y. Sun, T. Wang, S. Liu, J. Zhang, Y. Xue, and Y. Liu, "Combining fine-tuning and llm-based agents for intuitive smart contract auditing with justifications," in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE, 2025, pp. 330–342.

[26] Etherscan, "Uniswap v2 btc20 liquidity pool," https://etherscan.io/address/0xd50c5b8f04587d67298915e099e170af3cd6909a, 2024.

[27] ——, "Uniswap v2:router," https://etherscan.io/address/0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D#code, 2024.

[28] Github, "Supplementary material," https://github.com/AIS2Lab/DeFiScope/blob/main/supplementary_material.md, 2025.

[29] DeFiScope, "Test set used in fine-tuning," https://github.com/AIS2Lab/DeFiScope/blob/main/dataset/test_set.csv, 2025.

[30] R. McLaughlin, C. Kruegel, and G. Vigna, "A large scale study of the ethereum arbitrage ecosystem," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 3295–3312.

[31] BlockSec, "Elephantmoney analysis," https://x.com/BlockSecTeam/status/1513966074357698563, 2024.

[32] Immunefi, "Hack analysis: Cream finance oct 2021," https://medium.com/immunefi/hack-analysis-cream-finance-oct-2021-fc222d913fc5, 2024.

[33] Certik, "Journey of awakening incident analysis," https://www.certik.com/zh-CN/resources/blog/1YsQo8TnxCvwalqvtkFLtC-journey-of-awakening-incident-analysis, 2024.

[34] SlowMist, "Slow mist: Analysis of harvest.finance's hacked event," https://slowmist.medium.com/slow-mist-analysis-of-harvest-finances-hacked-event-63450b49e6a5, 2024.

[35] Etherscan, "Contract of convex finance," https://etherscan.io/address/0xd600A3E4F57E718A7ad6A0cbb10c2A92c57827e6, 2024.

[36] Ethereum, "Ethereum," https://ethereum.org/en/, 2024.

[37] B. S. Chain, "Binance smart chain," https://www.bnbchain.org/en/smartChain, 2024.

[38] DeFiLlama, "Defillama: All chains," https://defillama.com/chains, 2024.

[39] QuickNode, "Quicknode," https://www.quicknode.com, 2024.

[40] SunWeb3Sec, "Defihacklabs," https://github.com/SunWeb3Sec/DeFiHackLabs, 2024.

[41] N. L. Franklin, "Openleverage attack analysis," https://x.com/0xNickLFranklin/status/1774727539975672136, 2024.

[42] BlockSec, "Yield attack analysis," https://x.com/Phalcon_xyz/status/1782966561726156945, 2024.

[43] zokyo, "Safemoon attack analysis," https://x.com/zokyo_io/status/1641014520041840640, 2024.

[44] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, "The curious case of neural text degeneration," *arXiv preprint arXiv:1904.09751*, 2019.

[45] Z. Chen, S. M. Beillahi, and F. Long, "Flashsyn: Flash loan attack synthesis via counter example driven approximation," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[46] N. H. Bosi Zhang, X. Hu, K. Ma, and H. Wang, "Following devils' footprint: Towards real-time detection of price manipulation attacks," 2025.

[47] Q. McNemar, "Note on the sampling error of the difference between correlated proportions or percentages," *Psychometrika*, vol. 12, no. 2, pp. 153–157, 1947.

[48] Ancilia, "Inuko finance exploit analysis," https://medium.com/@Ancilia/inuko-finance-exploited-and-lost-60k-bond-89a285221e33, 2024.

[49] B. Phalcon, "Ark attack analysis," https://x.com/Phalcon_xyz/status/1771728823534375249, 2024.

[50] Inspex, "Inverse finance's incident analysis," https://inspexco.medium.com/inverse-finances-incident-analysis-inv-price-manipulation-b15c2e917888, 2024.

[51] Knownsec, "Memestake deflation model lightning loan attack analysis," https://medium.com/@Knownsec_Blockchain_Lab/memestake-deflation-model-lightning-loan-attack-analysis-8706b59bc9da, 2024.

[52] Ancilia, "Startwallets attack analysis," https://x.com/AnciliaInc/status/1781102805010550911, 2024.

[53] BlockSec, "Swapx attack analysis," https://x.com/BlockSecTeam/status/1630111965942018049, 2024.

[54] V. Finance, "On the vesper lend beta / rari fuse pool exploit," https://medium.com/vesperfinance/on-the-vesper-lend-beta-rari-fuse-pool-23-exploit-9043ccd40ac9, 2024.

[55] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig, "Pal: Program-aided language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 10 764–10 799.

[56] crytic, "Slither: the smart contract static analyzer," https://github.com/crytic/slither, 2024.

[57] N. Grech, L. Brent, B. Scholz, and Y. Smaragdakis, "Gigahorse: thorough, declarative decompilation of smart contracts," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1176–1186.

[58] N. Grech, S. Lagouvardos, I. Tsatiris, and Y. Smaragdakis, "Elipmoc: Advanced decompilation of ethereum smart contracts," *Proceedings of*

the ACM on Programming Languages, vol. 6, no. OOPSLA1, pp. 1–27, 2022.
[59] M. Suiche, "Porosity: A decompiler for blockchain-based smart contracts bytecode," *DEF con*, vol. 25, no. 11, 2017.
[60] OpenAi, "Openai pricing," https://openai.com/api/pricing/, 2024.
[61] Ancilia, "Ancilia," https://x.com/AnciliaInc, 2024.
[62] Beosin, "Beosin alert," https://x.com/BeosinAlert, 2024.
[63] BlockSec, "Blocksec phalcon," https://x.com/Phalcon_xyz, 2024.
[64] SlowMist, "Slowmist team," https://x.com/SlowMist_Team, 2024.
[65] BscScan, "Bscscan," https://bscscan.com, 2024.
[66] EtherScan, "Etherscan," https://etherscan.io, 2024.
[67] OKLink, "Oklink," https://www.oklink.com, 2024.
[68] Huggingface, "Phi-3-medium-128k-instruct," https://huggingface.co/microsoft/Phi-3-medium-128k-instruct, 2025.
[69] DeFiScope, "Detailed comparison results," https://github.com/AIS2Lab/DeFiScope/blob/main/model_comparison_result.md, 2025.
[70] ——, "Fine-tuned phi-3 model," https://huggingface.co/RocketRaccoonnn/Phi-3-medium-128k-instruct_LoRA_CASUAL_LM_lora_v2, 2025.
[71] Polygon, "Polygon," https://polygon.technology, 2025.
[72] Arbitrum, "Arbitrum," https://arbitrum.io, 2025.
[73] Z. Chen, Y. Liu, S. M. Beillahi, Y. Li, and F. Long, "Demystifying invariant effectiveness for securing smart contracts," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 1772–1795, 2024.
[74] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, "High-frequency trading on decentralized on-chain exchanges," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 428–445.
[75] K. Qin, L. Zhou, B. Livshits, and A. Gervais, "Attacking the defi ecosystem with flash loans for fun and profit," in *International conference on financial cryptography and data security*. Springer, 2021, pp. 3–32.
[76] K. Qin, L. Zhou, and A. Gervais, "Quantifying blockchain extractable value: How dark is the forest?" in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 198–214.
[77] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," in *2020 IEEE symposium on security and privacy (SP)*. IEEE, 2020, pp. 910–927.
[78] L. Zhou, X. Xiong, J. Ernstberger, S. Chaliasos, Z. Wang, Y. Wang, K. Qin, R. Wattenhofer, D. Song, and A. Gervais, "Sok: Decentralized finance (defi) attacks," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2444–2461.
[79] M. Zhang, X. Zhang, Y. Zhang, and Z. Lin, "{TXSPECTOR}: Uncovering attacks in ethereum from transactions," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 2775–2792.
[80] S. So, S. Hong, and H. Oh, "{SmarTest}: Effectively hunting vulnerable transaction sequences in smart contracts through language {Model-Guided} symbolic execution," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1361–1378.
[81] L. Zhou, K. Qin, A. Cully, B. Livshits, and A. Gervais, "On the just-in-time discovery of profit-generating transactions in defi protocols," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 919–936.
[82] B. He, Y. Chen, Z. Chen, X. Hu, Y. Hu, L. Wu, R. Chang, H. Wang, and Y. Zhou, "Txphishscope: Towards detecting and understanding transaction-based phishing on ethereum," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 120–134.
[83] K. Qin, S. Chaliasos, L. Zhou, B. Livshits, D. Song, and A. Gervais, "The blockchain imitation game," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 3961–3978.
[84] Y. Sun, D. Wu, Y. Xue, H. Liu, H. Wang, Z. Xu, X. Xie, and Y. Liu, "Gptscan: Detecting logic vulnerabilities in smart contracts by combining gpt with program analysis," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.
[85] Y. Gai, L. Zhou, K. Qin, D. Song, and A. Gervais, "Blockchain large language models," *arXiv preprint arXiv:2304.12749*, 2023.
[86] Y. Sun, D. Wu, Y. Xue, H. Liu, W. Ma, L. Zhang, M. Shi, and Y. Liu, "Llm4vuln: A unified evaluation framework for decoupling and enhancing llms' vulnerability reasoning," *arXiv preprint arXiv:2401.16185*, 2024.
[87] Y. Liu, Y. Xue, D. Wu, Y. Sun, Y. Li, M. Shi, and Y. Liu, "Propertygpt: Llm-driven formal verification of smart contracts through retrieval-augmented property generation," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2025.