

# Generic Adversarial Smart Contract Detection with Semantics and Uncertainty-Aware LLM

Yating Liu, Xing Su, Hao Wu, Sijin Li, Yuxi Cheng, Fengyuan Xu\*, Sheng Zhong  
National Key Lab for Novel Software Technology, Nanjing University, Nanjing, Jiangsu, China  
Emails: {yatingliu, xingsu, sijinli, yuxicheng}@mail.nju.edu.cn, {hao.wu, fengyuan.xu, zhongsheng}@nju.edu.cn

**Abstract**—Adversarial smart contracts, mostly on EVM-compatible chains like Ethereum and BSC, are deployed as EVM bytecode to exploit vulnerable smart contracts typically for financial gains. Detecting such malicious contracts at the time of deployment is an important proactive strategy preventing loss from victim contracts. It offers a better cost-benefit than detecting vulnerabilities on diverse potential victims. However, existing works are not generic with limited detection types and effectiveness due to imbalanced samples, while the emerging LLM technologies, which show its potentials in generalization, have two key problems impeding its application in this task: hard digestion of compiled-code inputs, especially those with task-specific logic, and hard assessment of LLMs’ certainty in their binary answers, i.e., yes-or-no answers. Therefore, we propose a generic adversarial smart contracts detection framework FinDet, which leverages LLMs with two enhancements addressing above two problems. FinDet takes as input only the EVM-bytecode contracts and identifies adversarial ones among them with high balanced accuracy. The first enhancement extracts concise semantic intentions and high-level behavioral logic from the low-level bytecode inputs, unleashing the LLM reasoning capability restricted by the task input. The second enhancement probes and measures the LLM uncertainty to its multi-round answering to the same query, improving the LLM answering robustness for binary classifications required by the task output. Our comprehensive evaluation shows that FinDet achieves a BAC of 0.9223 and a TPR of 0.8950, significantly outperforming existing baselines. It remains robust under challenging conditions including unseen attack patterns, low-data settings, and feature obfuscation. FinDet detects all 5 public and 20+ unreported adversarial contracts in a 10-day real-world test, confirmed manually.

## I. INTRODUCTION

While blockchain technology is advancing rapidly, its decentralized and anonymous nature also creates opportunities for malicious activities. In 2024, security incidents led to losses exceeding \$2.01 billion [29]. To protect the ecosystem, prior research mainly focused on detecting vulnerable smart contracts. However, many reported vulnerabilities are ultimately unexploitable [22]. Moreover, defenders face an inherent asymmetry, as they must secure against all threats, whereas attackers need only a single successful exploit. Other approaches monitor adversarial transactions in public mem-pools, but they cannot observe transactions in private pools and can only analyze malicious behavior after it has been broadcast or included on-chain.

To address these limitations, recent research efforts have shifted toward adversarial contract detection. Adversarial contracts are deliberately crafted smart contracts that exploit

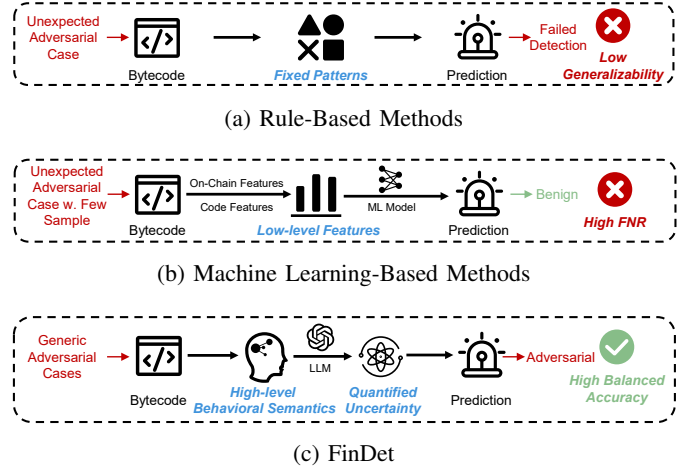


Fig. 1: Comparison of adversarial contract detection methods. (a) Rule-based methods rely on fixed patterns and fail on unexpected attacks. (b) ML-based methods leverage low-level features but struggle with few-shot or unseen cases, resulting in high FNR. (c) FinDet utilizes high-level semantics with LLM reasoning to achieve robust detection of generic adversarial contracts, achieving high balanced accuracy.

known vulnerabilities or manipulate protocol logic for illicit gain. These approaches focus on contracts that are both exploitable and likely to be deployed in real attacks. By identifying these high-impact threats ahead of execution, defenders can intervene before damage occurs and significantly reduce the burden of manual auditing.

Most adversarial contracts exist only as bytecode to conceal their malicious intent. As shown in Fig. 1, rule-based methods [43, 44] detect adversarial behaviors by disassembling bytecode and extracting control and data flows, which are then analyzed using manually defined heuristics and rule-based logic. However, these methods are effective only for single attack types and lack the flexibility to generalize beyond predefined patterns. To enable detection of multiple attack types, Machine learning (ML)-based methods [24, 38] leverage low-level intermediate representation (IR) and on-chain features to train detection models. However, these methods suffer from high false negative rates, reaching 51.00% for Lookahead [24] and 23.50% for Skyeye [38]. Moreover, they face two fundamental limitations: limited generalization ability, which impedes the detection of unseen attack patterns as adversarial behaviors continuously evolve beyond the training data, and

\*Fengyuan Xu is the corresponding author.

a strong dependence on large amounts of labeled data, which is often scarce and costly, resulting in significant performance degradation when training samples are insufficient.

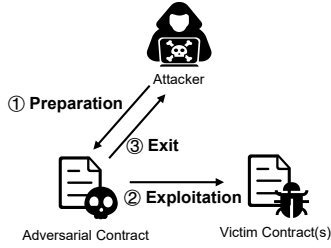


Fig. 2: Three-phase attack life-cycle of adversarial contracts.

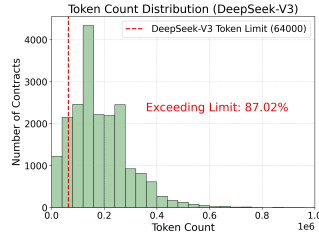


Fig. 3: Token distribution of TAC generated by Elipmoc [9] on DeepSeek-V3 (64K).

Encouragingly, large language models (LLMs) have demonstrated tremendous potential in understanding advanced code representations (e.g., source code), exhibiting strong capabilities in capturing behavioral semantics and interpreting smart contract logic. These strengths make LLMs a promising alternative for detecting adversarial smart contracts, especially in scenarios where labeled training data is scarce and costly to obtain. However, this approach faces several key challenges:

- **Challenge 1: Limited Understanding of Low-Level Intermediate Representations.** LLMs are primarily pre-trained on high-level source code, and thus lack the capability to accurately understand the low-level semantics of EVM bytecode. One alternative approach is to lift bytecode into IR, such as using Elipmoc [9] to transform bytecode into three-address code (TAC) for improved interpretability. However, as shown in Fig. 3, 87.02% of TAC representations exceed typical LLM input limits, leading to frequent detection failures. Fine-tuning LLMs directly on bytecode is possible but costly and limited by scarce labeled data.
- **Challenge 2: Diverse and Evolving Attack Strategies.** Adversarial contracts employ a wide range of attack methods that continuously evolve over time. Relying solely on superficial features of previously known attacks for decision-making limits the ability to detect unseen and emerging attack types effectively.
- **Challenge 3: Hallucination and Challenging Uncertainty Assessment.** LLMs tend to lose focus when processing complex logic, often resulting in high false positives due to conservative decisions under uncertainty [33]; since most mainstream LLMs are closed-source and operate as black-box systems, detecting and quantifying their hallucinations is particularly challenging and reduces detection reliability.

In this work, to leverage the capabilities of LLMs while addressing the above challenges, we propose FinDet, a generic adversarial smart contract detection method that enhances high-level behavioral semantics and enables quantified assessment of LLM uncertainty. FinDet relies solely on bytecode thus enabling early detection during the pre-deployment phase, and is also capable of robustly identifying previously unseen attack patterns. It operates in two stages. In Stage I, the

bytecode is lifted into a semi-structured natural language description, based on which we perform general-purpose analysis from the perspective of the contract’s overall semantics, and attack-specific analysis grounded in the semantics of operational logic. In Stage II, we conduct fine-grained uncertainty probing and assessment. By leveraging the multi-perspective semantic information obtained in Stage I through targeted questions and performing reliable fusion, FinDet achieves robust detection results.

To address **Challenge 1**, FinDet adopts a bytecode lifting mechanism that translates raw bytecode into semi-structured natural language descriptions. This process preserves critical semantic information while enhancing interpretability. Unlike traditional intermediate-representation-level static analyses that struggle to capture subtle contract logic, this semantic elevation enables more accurate and robust detection of adversarial behaviors. **Challenge 2** stems from the complexity and rapid evolution of adversarial attack strategies, making generic detection difficult. To tackle this, we analyzed many adversarial cases and identified common real-world attack patterns [5, 18, 25] following three stages: ① *Preparation*, ② *Exploitation* and ③ *Exit* (see Fig. 2 and **Observation 2**). FinDet systematically analyzes fund-flow reachability across these stages, capturing fundamental operational logic that differentiates adversarial from benign contracts. As to **Challenge 3**, we first transform the yes-or-no detection query to LLMs into a task of asking LLMs to assign probabilities among four levels of uncertainty, and then repeats the same task with disturbance prompts to check how certain LLMs are. Uncertainty in LLMs’ answering is captured by the entropy in those probability distributions [27], which is further used to derive with high confidence what the yes-or-no answer is in the end.

In our experiments, FinDet achieved a state-of-the-art balanced accuracy (BAC) of 0.9223, surpassing the baselines Lookahead and Skyeeye by 23.87% and 4.64%, respectively. It also attained a recall of 0.8950 on the adversarial class, outperforming Lookahead (82.65%) and Skyeeye (16.99%). Notably, FinDet consistently demonstrated robust performance under challenging scenarios, including unseen attack patterns, insufficient training data, and on-chain feature obfuscation, where baseline methods suffered significant performance degradation. During a 10-day real-world detection, FinDet successfully identified 30 adversarial contracts, among which 5 were previously undisclosed and exhibited clear adversarial intent. To support systematic evaluation and future research, we further curated the first dataset that categorizes adversarial contracts into normalized types, consisting of 200 adversarial and 20,000 benign samples.

We summarize our contributions as follows:

- We propose FinDet, the first generic and training-free detection framework that leverages the semantic understanding capabilities of LLMs to identify adversarial smart contracts directly from EVM bytecode, enabling detection in the pre-deployment phase. FinDet demonstrates robust performance under challenging scenarios such as unseen attack patterns and insufficient training data, and can seamlessly work with different LLMs without requiring fine-tuning.
- We enhance semantic understanding by integrating fund-

flow reachability analysis and multi-perspective semantic reasoning to capture fundamental adversarial patterns, both based on the lifted semi-structured natural language descriptions derived from bytecode.

- We design a novel quantitative uncertainty assessment of LLM outputs by extending labels to a fine-grained scale, incorporating multiple probing prompts, and applying entropy-based fusion to produce reliable uncertainty estimates.
- FinDet achieved state-of-the-art performance with a BAC of 0.9223 (surpassing the baselines by 23.87% and 4.64%) and an adversarial recall (i.e., TPR) of 0.8950 (exceeding the baselines by 82.65% and 16.99%). Furthermore, FinDet identified 29 previously undisclosed adversarial contracts during real-world detections.

## II. BACKGROUND

### A. Decentralized Finance

Blockchain serves as a decentralized, peer-to-peer, and verifiable infrastructure that records and secures transactions through distributed consensus while preserving user anonymity. Built on top of blockchains, Decentralized Finance (DeFi) enables permissionless and composable financial services via smart contracts. The core components of DeFi are accounts and transactions, which define ownership and trigger contract execution.

**Accounts.** In Ethereum, there are two types of accounts: Externally Owned Accounts (EOAs), which are controlled by private keys, and smart contract accounts, which are governed by code deployed on the blockchain.

**Transactions.** Transactions serve as the foundation of DeFi execution, with external transactions initiated by EOAs and internal transactions generated by smart contracts to carry out protocol logic.

### B. Vulnerabilities and Adversarial Contracts

Smart contracts are autonomous programs that manage assets and enforce logic in DeFi protocols, characterized by immutability, composability, and permissionless interactions. Vulnerabilities [26] are unintended weaknesses or flaws in smart contract code that can be exploited to cause incorrect behavior or asset loss, such as reentrancy or price manipulation. In contrast, adversarial contracts [43] are deliberately crafted malicious contracts designed to exploit known vulnerabilities or manipulate protocol logic for illicit gain.

### C. Large Language Models for Blockchain Security

The emergence of LLMs marks a major advance in language understanding and generation. Built on Transformer architectures with billions of parameters, they exhibit strong reasoning and semantic comprehension. In blockchain, LLMs have gained attention for tasks like smart contract auditing [15, 33], on-chain transaction analysis [11], and dynamic contract analysis [32]. These applications highlight LLMs' potential as powerful tools to enhance blockchain security.

## III. MOTIVATION, OBSERVATIONS AND THREAT MODEL

### A. Motivation Example

Fig. 4 illustrates a typical scenario of an attack via flashloans.

The blue arrows indicate the attack steps: (1) The attacker calls the `exploit` function, obtaining a flash loan from the DPP contract to secure initial capital; (2) A callback is triggered during the loan, invoking `DPPFlashLoanCall`, a function controlled by the attacker; (3-4) The attacker performs repeated token buy/sell operations to exploit price discrepancies; (5) The borrowed funds are then repaid to the lending protocol, and (6) the extracted profit is transferred to the attacker.

The yellow arrows illustrate the fund-flow throughout the attack: (a) a flash loan is issued from the DPP contract to the adversarial contract; (b) the adversary initiates a buy operation with a small input to artificially inflate the token price; (c) this is followed by a sell operation to extract profit based on the manipulated price; (d) the borrowed funds are then repaid to the DPP contract; and finally, (e) the remaining profit is transferred from the adversarial contract to the attacker's address. This motivating example reveals typical behavioral patterns of adversarial contracts. We elaborate on these observations in the following section.

### B. Observations

After identifying the core challenges, we further make the following key observations from our empirical analysis, which directly inform the design of our system.

**Observation 1: Long-Form Inputs Cause Loss of Contextual Focus.** We find that LLMs struggle to maintain semantic coherence when presented with entire contracts as monolithic inputs. Instead of capturing the overarching behavioral intent, the model often fixates on isolated or syntactically unusual fragments, leading to erroneous reasoning. For example, as shown in Fig. 16 in Appendix F, a benign contract was misclassified due to its intricate logic structure and dense permission checks. This observation motivates our use of structured prompting and hierarchical summarization to preserve contextual clarity and guide the model's attention more effectively.

**Observation 2: Adversarial Fund Flows Follow a Structured Lifecycle.** Through large-scale analysis of adversarial contracts, we observe that attacks are fundamentally driven by illicit fund acquisition and consistently follow a three-phase lifecycle: ① **Preparation**, where the attacker obtains necessary permissions or capital (e.g., via privilege escalation or flash loans); ② **Exploitation**, where the attacker manipulates contract logic or state to generate unauthorized gains; and ③ **Exit**, where profits are extracted through abnormal fund flows or obfuscation (see Fig. 2). This lifecycle reflects the intrinsic goal of financial exploitation, as fundamentally any contract attempting to steal funds must progress through these stages. It provides a principled abstraction of attack semantics and underpins our phase-aware modeling of fund-flow logic as a robust foundation for detecting adversarial behavior.

**Observation 3: Ambiguous Patterns Trigger Conservative Reasoning.** When encountering gray-area behaviors

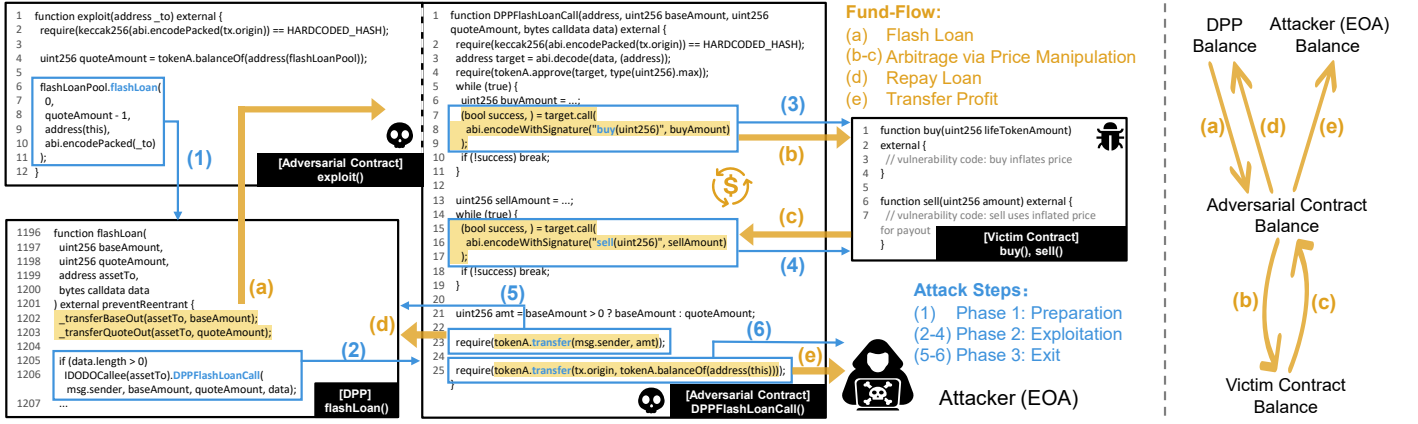


Fig. 4: Decompiled Solidity snippet of an adversarial contract involved in a flash loan attack on April 26, 2025. This case illustrates a typical flash loan exploit, following the three-phase fund-flow pattern commonly observed in adversarial contracts. The right part of the figure visualizes the corresponding fund transfers across different addresses and contracts during the attack.

such as redundant access checks, complex fallback logic, or reentrant fund management, LLMs tend to default to conservative classifications due to uncertainty, often leading to false positives. These behaviors may be essential for security or robustness in benign contracts but are misinterpreted as obfuscation or adversarial intent. This insight underscores the need to quantify model uncertainty, leading to our entropy-based scoring and fusion mechanism, which interprets model outputs as graded confidence signals rather than binary decisions.

### C. Threat Model, Scope, and Assumptions

FinDet targets adversarial contracts crafted to exploit on-chain vulnerabilities. We assume attackers have full access to public information, including victim contract source or decompiled code, and can deploy arbitrary contracts, send transactions, and use private channels (e.g., flashbots) to hide activity.

Attacks unrelated to contract-level vulnerabilities (e.g., private key leaks or off-chain compromises) and those by privileged users (e.g., rug pulls) are out of scope. We also exclude attacks solely via malicious transactions without a deployed adversarial contract.

Detection occurs pre-deployment (mempool stage), relying only on bytecode without runtime blockchain data. Our approach generalizes by analyzing semantic intent rather than brittle signatures or handcrafted datasets. A detection is considered successful if malicious behavioral indicators are identified when the contract is prepared for deployment, thereby allowing the contract to be correctly classified as adversarial.

## IV. OVERVIEW OF FINDET

This section presents the overall design of FinDet, a general framework that leverages the strong semantic understanding capabilities of LLMs to detect adversarial smart contracts directly from EVM bytecode. FinDet supports generic detection and is capable of identifying previously unseen attack patterns. As illustrated in Fig. 5, the framework consists of two main stages: high-level behavioral semantic analysis (Stage I) and quantified uncertainty with probing and fusion (Stage II). We

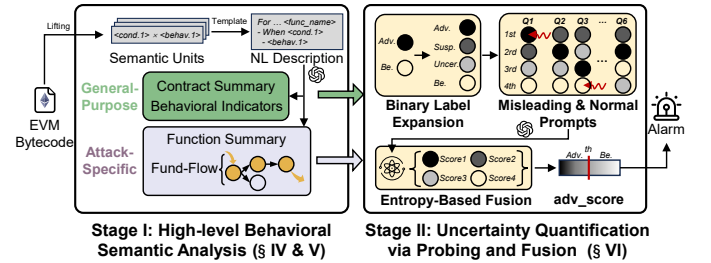


Fig. 5: A high-level workflow of FinDet.

parallelize LLM queries within Stage I and Stage II separately to improve efficiency.

We begin by lifting raw EVM bytecode into semantic units and transforming them into semi-structured natural language description through templates (§IV-A) to overcome the limited understanding of low-level intermediate representations (**Challenge 1**). We perform general-purpose analysis (§IV-B) to understand the overall purpose of the contract and extract sensitive semantics. We also conduct attack-specific analysis (§IV-C) by examining each function individually to summarize its intent, suspicious behaviors, and supporting evidence. Specifically, to address the diverse and evolving attack strategies (**Challenge 2**), we conduct fund-flow reachability analysis (§V) based on the common three-phase adversarial pattern shown in Fig. 2. This approach traces attacker-controlled fund-related operations and uncovers potential exploitation paths. To address LLM hallucination and the difficulty of assessing uncertainty, especially in black-box scenarios (**Challenge 3**), we introduce a probing- and fusion-based method that extends binary classification into fine-grained risk assessment (§VI). Leveraging normal and misleading probes with entropy-based fusion, we quantify and rank LLM uncertainty to achieve reliable detection.

### A. Natural Language Description Generation

As shown in Fig. 3, converting bytecode into TAC via Elipmoc [9] makes over 87% of contracts exceed typical LLM



### Prompt Template of Stage I

**System:** You are an excellent smart contract attack detector. You are given natural language descriptions of smart contracts and are skilled at identifying malicious or suspicious intent. Your task is to generate concise and structured reports highlighting attacker-like behaviors.

#### ### Task: function\_summary ###

Please analyze each described function individually, focusing on attacker-like or abnormal behaviors. For each function, provide:

- 1) Function Purpose: A short summary of the function's behavior (1-2 sentences).
- 2) Suspicious Behavior: Does the function contain signs of attacker-like or exploitative behavior? (Yes/No)
- 3) Evidence (if any): Explain why it might be malicious or abnormal (1-2 sentences).

#### Response Format (strictly follow):

[Function: <function\_name>  
Purpose: <summary>  
Suspicious Behavior: <Yes or No>  
Evidence: <brief explanation>

#### ### Task: contract\_summary ###

Based on the above analysis, please perform a high-level analysis of the smart contract based on its natural language description. Your response must include: briefly summarize the overall purpose and behavior of the contract (40-80 words).

#### Response Format (strictly follow):

[Contract Summary]  
<Your summary here>

#### ### Natural Language Description of Smart Contract ###

[NL description of the smart contract.]

Fig. 6: The prompt for stage I

token limits, highlighting the need for more compact, high-level inputs. Unlike Disco [31] that convert semi-structured natural language back into source code, we reason directly over its semantic content. We retain the natural language description for three reasons.

First, code regeneration may lead to hallucinations and inconsistencies. In fact, only 28.5% of the decompiled contracts in [31] are semantically equivalent to the original versions. Second, code generation increases token usage and latency, while alternatives like three-address code are too verbose for long bytecode sequences (see Fig. 2). Third, natural language provides a concise, high-level abstraction that aids efficient downstream analysis. All components in our framework operate directly on this representation.

### B. General-Purpose Analysis

General-purpose analysis strategy begins with understanding the contract's overall intent and core functionality. By capturing this high-level perspective, we can more effectively interpret the contract's design objectives and evaluate its potential security risks. To achieve this, we prompt a LLM to synthesize information across all functions and generate a comprehensive, contract-level summary, as illustrated in Fig. 6. An example summary for the contract depicted in Fig. 4 is provided in Fig. 14 in Appendix E. Furthermore, we extract several contract-level indicators from the natural language descriptions, including the number and proportion of external calls, unknown functions, and bot-related functions, as well

as the presence of fund-transfer behaviors (e.g., operations indicative of attempts to drain the contract balance). These metrics contribute to a holistic understanding of the contract's structural characteristics and its potential security implications.

### C. Attack-Specific Analysis

To enable fine-grained analysis of adversarial contracts, we adopt a function-level behavioral perspective that guides the LLM to examine each function's intended purpose, suspicious behaviors, and supporting evidence. This approach reveals subtle signs of malicious logic that may be overlooked in global-level summaries. While benign contracts typically consist of modular functions with well-defined objectives and appropriate access control, adversarial contracts often embed sensitive operations or privilege escalation within ostensibly benign functions. Accurately identifying each function's intent is thus essential for distinguishing malicious behavior from legitimate functionality. Moreover, detecting suspicious patterns, such as unsafe external calls or attacker-controlled inputs, and extracting concrete evidence helps preserve critical indicators. As shown in Fig. 13, this function-centric analysis is based on the contract presented in Fig. 4 and guides the LLM to develop a detailed understanding of the contract's behavior. We also retain the natural language descriptions of functions without identified names, as these are more likely to contain malicious logic.

## V. FUND-FLOW REACHABILITY ANALYSIS

Our key insight is that most adversarial contracts are fundamentally driven by the goal of illegitimately acquiring funds. This insight is supported by two common characteristics: (1) Economic Incentives: Adversarial contracts are fundamentally driven by the goal of illicit financial gain, which leads to deliberate manipulation of fund flows within the contract. (2) Design Differences: While benign contracts typically enforce fund transfers through rigid and auditable logic, such as owner-only withdrawals or strict conditional checks, adversarial contracts introduce flexible or hidden pathways, enabling unauthorized fund movement. These characteristics highlight the necessity of fund-flow analysis to effectively distinguish malicious behaviors.

### A. Control Dependency Forest Construction

As shown in Fig. 7 (from (a) to (d)), we propose a control dependency forest to systematically analyze the semantic structure of smart contracts. Given a natural language description of a contract (a), we first perform ❶ chunking to segment it into a set of function-level descriptions (b), where each chunk corresponds to the natural language description of a single function. For each function, we then ❷ construct a control dependency tree by capturing the control dependencies expressed in the description. In each tree, every sentence is treated as a node, with the function signature serving as the root. Each node in the tree is heuristically classified into one of four categories:

- function: the root node representing the function signature;

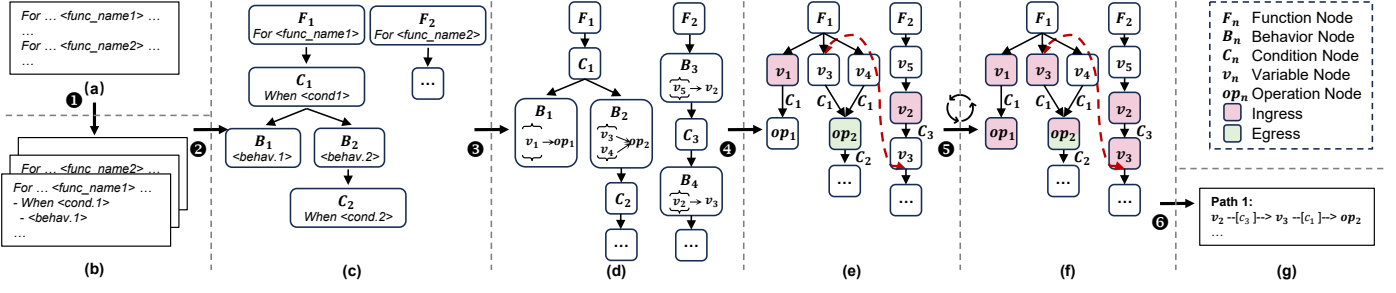


Fig. 7: Workflow of fund flow reachability analysis consisting of six steps: ❶ chunking, ❷ contract forest construction, ❸ behavior node entity resolution, ❹ graph transformation & ingress/egress identification, ❺ fund flow reachability analysis, and ❻ reverse pruning & output.

- **behavior**: nodes that describe functional actions, such as external calls, assignments, and delegate calls;
- **condition**: nodes that express conditional logic, such as if-else branches, require statements, and loops;
- **unknown**: fallback nodes for sentences that cannot be reliably parsed, serving as a safety mechanism.

This classification is feasible because the natural language descriptions are generated using a predefined template, ensuring consistent structure and phrasing. As a result, the entire contract is transformed into a control dependency forest (c).

We further ❸ refine the behavior node set  $\mathcal{N}_{\text{behavior}} = \{B_1, B_2, \dots, B_m\}$  by assigning each node a fine-grained behavior type based on its semantic role (e.g., external call, assignment, etc.). The specific behavior types and their corresponding processing rules are summarized in TABLE I. For each behavior node  $B_i \in \mathcal{N}_{\text{behavior}}$ , we extract the involved entities, including variables  $v$  and function operations  $op$ , and define a propagation tuple as follows:

$$B_i = (\mathcal{X}_{\text{src}}, x_{\text{dst}}) \quad (1)$$

where  $\mathcal{X}_{\text{src}}$  denotes the set of source entities (i.e., inputs that influence the behavior), and  $x_{\text{dst}}$  refers to the single destination entity (i.e., the output whose value may be determined by the sources). Both  $\mathcal{X}_{\text{src}}$  and  $x_{\text{dst}}$  are filtered to exclude constants such as numeric literals or hardcoded addresses. To distinguish homonymous entities across different functions, we prepend the function name as a namespace to local entities (e.g., `pancakecall:param1` and `transfer:param1` refer to distinct entities). Global entities retain their original names without such prefixes (e.g., `msg.caller`). Depending on the behavior type, either the source set or the destination field (or both) in the propagation tuple may be empty. For all other 3 node types, we preserve their original natural language descriptions without further parsing, in order to maintain the full semantic context of each function. As a result, we obtain the refined forest of behavior nodes shown in Fig. 7 (d).

### B. Graph Transformation

As illustrated in Fig. 7 (from (d) to the transformed graph in (e)), we perform ❹ graph transformation first to convert the contract forest into a structure suitable for fund

flow reachability analysis. This process relies on previously extracted entity information from behavior nodes. Specifically, for each function tree, we perform a depth-first search (DFS) starting from the root function node  $F_i$ . During the traversal, we maintain a set of visited entities  $e$  and their associated condition sets  $\mathcal{C}$ , denoted as  $(e, \mathcal{C})$ . This set is initialized with the parameters of  $F_i$  and global variables, each associated with an empty condition set.

For each behavior node  $B_i$  encountered during traversal, we extract its corresponding source entities  $\mathcal{X}_{\text{src}}^{(i)}$  and destination entity  $x_{\text{dst}}^{(i)}$  from Eq. (1). For each  $x_j \in \mathcal{X}_{\text{src}}^{(i)}$ , if  $x_j$  is already in the visited set with condition set  $\mathcal{C}_j$ , and  $x_{\text{dst}}^{(i)}$  has not yet been visited, we identify the set of condition nodes  $\mathcal{C}_k$  that have appeared along the current traversal path since  $x_j$  was last added. If  $\mathcal{C}_k$  is non-empty, we construct a new edge from  $x_j$  to  $x_{\text{dst}}^{(i)}$ , annotated with the combined condition set  $\mathcal{C}_j \cup \mathcal{C}_k$ , denoted as:

$$x_j \xrightarrow{\mathcal{C}_j \cup \mathcal{C}_k} x_{\text{dst}}^{(i)}$$

We then add  $x_{\text{dst}}^{(i)}$  to the visited set with its updated condition set  $\mathcal{C}_j \cup \mathcal{C}_k$ . Moreover, as previously mentioned, we normalize entity names during behavior node entity resolution: global variables retain their original names, while local variables are prefixed with the name of the function to which they belong, serving as their namespace. This enables us to identify the same entity across different functions through simple name matching. Based on this process, we supplement inter-function edges by connecting nodes that refer to the same entity, such as the variable `v3` in both  $F_1$  and  $F_2$  in Fig. 7 (e). In this process, we exclude all unknown nodes and any behavior nodes that do not yield a resolvable entity.

Through this process, we obtain a transformed graph where all nodes (except the root  $F_i$ ) correspond to entities extracted from behavior nodes, including both variables and operations. All condition nodes are encoded as edge annotations, as shown in Fig. 7 (e).

### C. Ingress and Egress Identification.

To facilitate fund-flow reasoning in adversarial contract analysis, we first identify potential entry and exit points of value-related information within smart contracts. These points, termed as ingress and egress, respectively, mark where

TABLE I: Behavior types and propagation rules for fund-flow reachability. For each type, we define a source entity set  $\mathcal{X}_{\text{src}}$  and a destination entity  $x_{\text{dst}}$  to identify ingress and egress points. Only fund-relevant behaviors (e.g., attacker-controlled inputs, transfers, or external calls) are considered. Neutral behaviors such as returns, logs, or built-in calls are excluded as they do not affect fund flows.

Behavior Type	NL Description Template	$\mathcal{X}_{\text{src}}$	$x_{\text{dst}}$
assignment	it updates the state variable $x$ to $y$	$\{y\}$ (if not constant)	$x$
external call	it triggers the external call to <code>contract.function(<math>x_1, x_2, \dots, x_n</math>)</code>	$\{x_1, x_2, \dots, x_n\}$ (if not constant)	<code>contract.function</code>
delegate call	it delegates a call to <code>contract.function(<math>x_1, x_2, \dots, x_n</math>)</code>	$\{x_1, x_2, \dots, x_n\}$ or <code>calldata</code> (if present)	<code>delegatecall</code>
contract creation	it creates a new smart contract with creation code $c$ (and optional salt $s$ ), and gets a new address $a$	$\{s\}$ (optional, if not constant)	$a$
transfer	it transfers $v$ wei to $a$ (with gas $g$ )	$\{v\}$ (if not constant)	<code>transfer(<math>v</math>)</code>
return	it returns $x_1, x_2, \dots, x_n$	$\emptyset$	-
log emission	it emits the log event with parameter(s) $x_1, x_2, \dots, x_n$	$\emptyset$	-
built-in call	it calls a built-in function $f$	$\emptyset$	-
other	(fallback for unmatched patterns)	$\emptyset$	-

**Note:** In the case of *contract creation*, the variable  $s$  denotes the *salt*, which is an optional parameter influencing the new contract address. If not present, the source variable set is empty.

#### Algorithm 1: Fund-Flow Reachability Analysis

**Input:** Variable-level dependency graph  $G = (V, E)$  with edges labeled by conditions  
**Output:** Fund-Flow paths  $\mathcal{P}$  from ingress to egress

```

1  $\mathcal{S}_{\text{ingress}}, \mathcal{S}_{\text{egress}} \leftarrow \text{initial\_ingress\_egress}(\text{predefinedIngress}, \text{predefinedEgress}, \mathcal{F});$ 
2  $\mathcal{E}_{\text{visited}} \leftarrow \{(v, \emptyset) \mid v \in \mathcal{S}_{\text{ingress}}\};$  // Visited variables and associated conditions
3  $\text{updated} \leftarrow \text{true};$ 
4 while  $\text{updated}$  do
5    $\text{updated} \leftarrow \text{false};$ 
6   foreach  $v \in V$  such that  $v \in \text{Dom}(\mathcal{E}_{\text{visited}})$  do
7     foreach  $\text{edge } (v \xrightarrow{\text{conds}} u) \in E$  do
8       if  $u \notin \text{Dom}(\mathcal{E}_{\text{visited}})$  then
9          $\mathcal{E}_{\text{visited}} \leftarrow \mathcal{E}_{\text{visited}} \cup \{(u, \text{conds})\};$ 
10         $\text{updated} \leftarrow \text{true};$ 
11      end
12    end
13  end
14 end
15  $\mathcal{P} \leftarrow \text{ReverseDFSPrune}(G, \mathcal{E}_{\text{visited}}, \mathcal{S}_{\text{ingress}}, \mathcal{S}_{\text{egress}});$ 
16 return  $\mathcal{P}$ 
```

attacker-controlled inputs may enter the execution flow and where critical financial operations may be triggered. As illustrated in Fig. 7 (from (e) to (g)), these anchors play a foundational role in subsequent reachability analysis.

To ground this identification in real-world attack semantics, we compiled predefined sets of ingress and egress patterns based on their relevance to adversarial behaviors, corresponding to the three stages illustrated in Fig. 2, as shown in Listing 1.

Ingress points represent attacker-controllable or externally observable variables that may initiate value or permission flows. These include the function caller (`msg.sender`), the amount of Ether sent with the call (`msg.value`), the transaction initiator (`tx.origin`), and the contract's current balance (`address(this).balance`). In addition, we treat function parameters as contextual ingress points, since they often carry user-supplied input affecting internal logic and state. Collectively, these ingress points capture the primary channels

through which an attacker can inject external influence into a contract. Semantically, they correspond to the beginning of the ① *Preparation* phase of an attack, where the adversary gains capital, triggers a flash loan, or manipulates input to set up downstream exploitation.

Egress points, in contrast, denote critical financial or permission-related operations whose execution under attacker influence may lead to fund extraction, unauthorized transfers, or system compromise. These operations encompass both standard and low-level behaviors. On the standard side, functions like `transfer`, `transferFrom`, and `approve` are commonly used in ERC-20 tokens to move funds or grant allowances, and thus are directly tied to asset control. Functions such as `deposit` and `withdraw` are prevalent in DeFi protocols, where they govern asset custody and access. On the low-level side, `delegatecall` can lead to arbitrary code execution, while `selfdestruct` can irreversibly drain the contract balance. These egress points typically manifest during the ② *Exploitation* phase, where internal state is manipulated for gain, and the ③ *Exit* phase, where profits are withdrawn or laundered. By tracing propagation paths from ingress to egress, we aim to capture the semantic essence of adversarial fund flows.

By matching behavior node variables against these predefined patterns, we extract an ingress node set  $\mathcal{S}_{\text{ingress}}$  and an egress node set  $\mathcal{S}_{\text{egress}}$  from each function. These sets provide the semantic anchors for downstream fund-flow analysis and adversarial intent inference.

```

1 PREDEFINED_INGRESS = {
2   "msg.sender",      # function caller
3   "msg.value",       # incoming ETH value
4   "tx.origin",       # transaction origin
5   "address(this).balance", # contract's balance
6   "<function parameters>" # handled in each function
7 }
8 PREDEFINED_EGRESS = {
9   "transfer",        # token/native transfer
10  "transferFrom",    # delegated transfer
11  "approve",         # grant allowance
12  "deposit",         # asset deposit
13  "withdraw",        # asset withdrawal
14  "flashLoan",       # flash loan
15  "swapExactTokensForTokens", # token swap
16  "selfdestruct",    # destroy & transfer
17  "delegatecall"     # external logic call

```

Listing 1: Predefined ingress and egress for fund-flow-oriented adversarial analysis

#### D. Fund-Flow Reachability Analysis

Given a set of ingress points  $\mathcal{S}_{\text{ingress}}$ , we perform fund-flow propagation using a depth-first search (DFS) strategy, as outlined in Algorithm 1. Starting from each ingress node, we recursively explore downstream variables. When a variable is first encountered, it is added to the reachable set, and the search continues until no new nodes are discovered. To enable propagation across functions, we leverage the normalized naming scheme that identifies equivalent entities in different functions, allowing traversal through inter-function edges. To prevent infinite recursion caused by cycles, we maintain a visited set to skip already explored nodes.

After forward propagation, we perform a backward traversal from the set of egress points  $\mathcal{S}_{\text{egress}}$  to retain only semantically valid paths that originate from a predefined ingress and lead to a predefined egress. This step prunes irrelevant branches that do not contribute to the fund flow. For example, in Fig. 7 (f), although a path exists from  $v_1$  to  $op_1$ , it is discarded since  $op_1 \notin \mathcal{S}_{\text{egress}}$ . In contrast,  $op_2 \in \mathcal{S}_{\text{egress}}$ , and since  $v_3$  in both  $F_1$  and  $F_2$  refers to the same entity, a valid cross-function path  $v_2 \xrightarrow{c_3} v_3 \xrightarrow{c_1} op_2$  is preserved. Here, the intermediate conditions  $c_3$  and  $c_1$  are retained on the corresponding edges to encode control-flow context.

This analysis yields a set of fund-flow paths, each represented as a variable-level dependency chain annotated with the conditions encountered along the way. These conditions are expressed in natural language and are recorded for downstream reasoning, but are not enforced during traversal. Such a design balances semantic interpretability and scalability, facilitating subsequent analysis, including large language model-based reasoning.

## VI. UNCERTAINTY QUANTIFICATION VIA PROBING AND FUSION

LLMs are prone to overconfidence in their predictions. As demonstrated by our **Observation 2** (in §III-B), when LLMs are instructed to perform binary classification of smart contracts into adversarial or benign, they often assign overly confident labels to contracts that fall into a gray area of partial suspicion, misclassifying them as adversarial (see Baseline<sub>NL</sub> in §VII-C). This tendency directly increases the false positive rate in naive detection pipelines. A straightforward idea might be to request the LLM to output both the predicted label and its associated confidence. However, studies [42, 45] have shown that LLM-generated confidence scores are typically concentrated within the 80%-100% range and frequently rounded to multiples of 5. Although this pattern superficially resembles human-like uncertainty expression, it lacks proper probabilistic calibration and leads to outputs that are simultaneously overconfident and fragile, particularly in black-box scenarios where internal logits or attention distributions are inaccessible.

To address this issue, we propose a confidence-aware probing and fusion framework. Our approach begins by transforming the binary classification task into a graded confidence spectrum and normalizing verbalized confidence scores,

thereby enabling more fine-grained analysis. Subsequently, we employ both normal and misleading prompts to elicit diverse model outputs, and then perceive the variations in these outputs to probe the model’s uncertainty from multiple perspectives. Finally, we quantify uncertainty using entropy and perform a weighted fusion of information across multiple views. This comprehensive process leads to more robust predictions and enhanced detection performance. Specifically, our approach consists of the following steps:

#### A. Binary Label Expansion

In binary classification tasks, directly querying LLMs for confidence levels often leads to overly confident predictions. In contrast, comparative judgments based on relative rankings have been found to yield more reliable results [42]. To enable fine-grained confidence assessment, we extend the binary adversarial detection task into a four-level label spectrum. Specifically, we define the label space as:

$$\mathcal{L} = \{\text{adversarial}, \text{suspicion}, \text{uncertain}, \text{benign}\}.$$

Rather than relying on majority voting, our method focuses on generating a ranked list of labels along with their corresponding verbalized confidence scores, laying the foundation for subsequent quantitative analysis. Specifically, we prompt the LLM to rank the four label options by their likelihood and assign verbal confidence scores that sum to 100%, ensuring both interpretability and comparability. An illustrative example of this prompt is shown in Fig. 8. This structured output not only supports precise quantification of suspicion levels but can also be combined with the probing strategy introduced in the next section. By jointly leveraging ranked verbal confidence scores and consistency analysis of label orderings across multiple queries, we can better detect fluctuations in the LLM’s confidence. This combined approach enables more effective sensing of prediction uncertainty and leads to more robust classification results than relying on absolute scores alone.

#### B. Confidence Probing

In Stage I, we obtained both general and attack-specific information regarding the contract. Due to their abstract nature, general information often fail to capture concrete behavioral patterns, thereby biasing the model toward benign classifications. In contrast, attack-specific information provides detailed contextual cues more commonly associated with malicious behavior. However, such information may not always clearly indicate attacking behavior due to its specificity and may bias the model towards malicious judgment in some cases. Hence, incorporating both global and specific perspectives is essential for a more comprehensive behavioral analysis.

Furthermore, prior research has shown that misleading or weakly suggestive prompts can influence the model’s expressed confidence, even when the final prediction remains unchanged. For example, if the LLM is highly confident that a contract is adversarial, introducing a misleading cue implying benign behavior has minimal effect. Conversely, when the model is uncertain, a misleading cue suggesting adversarial intent can increase its confidence in classifying the contract as adversarial. Building on this insight, we propose the use of misleading prompts to probe the LLM’s confidence expression.



### Prompt Template of Stage II

[Different contract information: general / specific]

You are analyzing a smart contract deployed on the blockchain.

Provide your 4 best guesses and the probability that each is correct (0% to 100%) for the following question. Give your step-by-step reasoning in a few words first, and then give the final answer using the following format:

G1: <ONLY the option letter of first most likely guess; not a complete sentence, just the guess!>  
P1: <ONLY the probability that G1 is correct, without any extra commentary whatsoever; just the probability!>  
...  
G4: <ONLY the option letter of 4-th most likely guess; not a complete sentence, just the guess!>  
P4: <ONLY the probability that G4 is correct, without any extra commentary whatsoever; just the probability!>

The four options represent different levels of suspicion about whether the contract is an adversarial contract:

- (A) adversarial: clearly malicious, with strong evidence of harmful or exploitative behavior.
- (B) suspicion: likely malicious, showing noticeable but not definitive attack patterns.
- (C) uncertain: appears non-malicious but lacks enough evidence to be fully confident.
- (D) benign: clearly non-malicious, safe with no signs of harmful behavior.

Options: (A) adversarial (B) suspicion (C) uncertain (D) benign

(Optional) Hint: I think the answer should be {(A) adversarial/(D) benign}.

Fig. 8: Prompt Template of Stage II. The contract information of general-purpose analysis is provided in Fig. 11, and the attack-specific information is provided in Fig. 12.

Specifically, we design six types of prompts to evaluate the model's confidence across diverse perspectives, defined as:

$\mathcal{P} = \{\text{g\_normal}, \text{s\_normal}, \text{g\_mislead\_adv}, \text{g\_mislead\_be}, \text{s\_mislead\_adv}, \text{s\_mislead\_be}\},$

where `g_normal` and `s_normal` correspond to general and attack-specific information, respectively; `g_mislead_adv` and `g_mislead_be` denote adversarial and benign misleading prompts based on general information; while `s_mislead_adv` and `s_mislead_be` apply similar misleading cues to attack-specific information. Each misleading prompt contains a soft hint such as: “*Hint: I think the answer should be (A) adversarial / (D) benign*”, as illustrated in Fig. 8.

### C. Entropy-Based Fusion and Decision

We adopt an entropy-based approach to fuse multi-perspective outputs from 6 prompts, incurring virtually no time overhead. From the perspective of information theory, confidence entropy quantifies the uncertainty in the output distribution. A lower entropy indicates a more reliable prediction, consistent with Shannon's information theory [27]. If the LLM produces low-entropy confidence distributions under different prompts, it suggests higher output stability and reliability.

Each of the 6 prompts generates a confidence distribution over 4 labels, denoted as:

$$\left\{ (G_i^{(p)}, P_i^{(p)}) \mid p \in \mathcal{P}, i \in \mathcal{L}, \sum_{i \in \mathcal{L}} P_i^{(p)} = 100\% \right\}, \quad (2)$$

where  $G_i^{(p)}$  is the label  $i$  under the prompt  $p$ , and  $P_i^{(p)}$  is the corresponding confidence score, which sums to 100% across all labels for each prompt.

To quantify the uncertainty of each prompt's output, we compute the entropy as

$$H^{(p)} = - \sum_{i \in \mathcal{L}} p_i^{(p)} \log p_i^{(p)}, \quad (3)$$

where  $H^{(p)}$  denotes the entropy score corresponding to the prompt  $p$ .

A lower entropy implies a more confident prediction. Hence, we define the weight of prompt  $p$  as:

$$w^{(p)} = \frac{1}{H^{(p)} + \varepsilon}, \quad \varepsilon = 10^{-6}. \quad (4)$$

Each prompt also produces a ranked list over the 4 labels. We assign discrete scores to the labels based on their ranks, with the highest-ranked label receiving 3 points, followed by 2, 1, and 0. Let  $s_i^{(p)}$  denote the score assigned to label  $i \in \mathcal{L}$  under prompt  $p \in \mathcal{P}$ . The final weighted score for each label is computed as:

$$S_i = \sum_{p \in \mathcal{P}} w^{(p)} \cdot s_i^{(p)}. \quad (5)$$

We then normalize these scores to a probability distribution:

$$\hat{S}_i = \frac{S_i}{\sum_{j \in \mathcal{L}} S_j}. \quad (6)$$

To make a binary decision, we merge the four labels into two groups. We compute:

$$\text{adv\_score} = \hat{S}_{\text{adversarial}} + \hat{S}_{\text{suspicion}}, \quad (7)$$

$$\text{be\_score} = \hat{S}_{\text{benign}} + \hat{S}_{\text{uncertain}}. \quad (8)$$

If  $\text{adv\_score} > \text{be\_score}$ , the contract is classified as adversarial; otherwise, it is classified as benign.

### D. Adaptive Detection Trade-offs

Thanks to our flexible aggregation strategy, we obtain a continuous adversarial score (`adv_score`) instead of a simple binary classification result. This enables fine-grained perception of the LLM's confidence level in its decisions. Users may optionally select different adversarial score thresholds tailored to their specific needs. In scenarios where a low FNR is critical, a lower threshold can be employed to enhance detection sensitivity. Conversely, when the goal is to minimize the FPR, a higher threshold is recommended to reduce false alarms. Specifically, samples with scores exceeding the chosen threshold are classified as adversarial, whereas those below it are labeled benign. The corresponding trade-offs between sensitivity and specificity under different thresholds are illustrated in §VII-E2, with further quantitative details provided in the experimental section.

## VII. EVALUATION

### A. Experimental Setup

1) *Dataset*: To comprehensively evaluate FinDet, we constructed two datasets:  $\mathcal{D}_{GT}$ , a ground-truth-labeled dataset, and  $\mathcal{D}_{RW}$ , a collection of contracts from the wild.

For the adversarial contracts in  $\mathcal{D}_{GT}$ , due to the limited and fragmented availability of labeled adversarial contract data, we made our best effort to comprehensively collect relevant information and annotate it through a combination of original source labels and expert refinement. Specifically, we gathered data from five sources [3, 34, 38, 43, 44]. From these sources, we extracted vulnerability type labels based on the victim contracts involved in reported attack incidents and used these labels to annotate the corresponding adversarial contracts. We focus on attacks actively initiated by adversarial contracts, excluding cases such as rug pulls and private key compromises. To ensure consistent and accurate classification, we further normalized the vulnerability categories by referencing incident reports (e.g., tweets, blogs) and analyzing the decompiled pseudocode of the contracts. The final labels were aligned with the OWASP Smart Contract Top 10 (2025)<sup>1</sup>, a widely recognized guideline that highlights the most critical vulnerabilities in smart contracts. Finally, we collected 200 adversarial contracts with assigned categories, as summarized in TABLE IX. This represents the largest known dataset of adversarial contracts with explicitly labeled attack types to date. For the benign subset of  $\mathcal{D}_{GT}$ , we utilized the benign contracts provided by Lookahead and randomly sampled 20,000 contracts.

As for  $\mathcal{D}_{RW}$ , we collected contract data from BscScan over a 10-day period from April 18 to April 27 (UTC time), 2025. We extracted transaction information and contract creation records to obtain contract addresses, and subsequently crawled the corresponding runtime bytecode. Contracts that had already self-destructed were excluded from the dataset. As a result, we obtained 13,210 valid contracts. This dataset was only used in §VII-F.

2) *Metrics*: We use the Balanced Accuracy (BAC)<sup>2</sup> to evaluate detection performance. Balanced accuracy is particularly suitable for binary and multiclass classification problems [1, 6, 19] involving imbalanced datasets. It is defined as the average of recall values across all classes, and is particularly suitable for binary and multiclass classification problems involving imbalanced datasets and has been widely adopted in detection tasks on imbalanced datasets [39]. To provide a more comprehensive evaluation, we also report the False Positive Rate (FPR), False Negative Rate (FNR), True Positive Rate (TPR), and True Negative Rate (TNR).

3) *Baseline Selection*: To our knowledge, only two prior works perform adversarial contract detection directly from bytecode without targeting specific attack types:

- **Lookahead [24]**: Uses 40 handcrafted features (on-chain transactions and static bytecode features) and a PSCFT module to extract semantic information from bytecode. Both are inputs to machine learning classifiers.

- **Skyeye [38]**: Selects 12 features from Lookahead and applies control flow graph (CFG) segmentation for semantic extraction. These representations are also used with machine learning models. We reproduced their method from partially released code.

For fair comparison, we followed their experimental setups: oversampling adversarial contracts to address class imbalance, splitting the dataset 80/20 for training/testing, and using 10 epochs for Lookahead and 20 epochs for Skyeye.

4) *Implementation*: Experiments were conducted on an Ubuntu 24.04.1 server with a 64-core Intel Xeon Gold 6426Y CPU, eight NVIDIA GeForce RTX 4090 GPUs (24GB VRAM each), and 256GB RAM. Our method uses deepseek-v3 [14] as the default LLM. To evaluate generalizability, we also test claude-3-5-haiku-20241022 [2] and gpt-4o-2024-08-06 [21].

#### 5) Research Questions:

- **RQ1**: How does FinDet compare to existing baseline methods in detecting adversarial contracts under a training-free paradigm? (**Baseline Comparison**)
- **RQ2**: What is the impact of individual components in the Stage I probing module on detection performance? (**Ablation Study of Stage I**)
- **RQ3**: What is the impact of each component in the Stage II fusion and reasoning module? (**Ablation Study of Stage II**)
- **RQ4**: How efficient and adaptable is FinDet in terms of inference time, token cost, cross-LLM generalization, and threshold flexibility? (**Scalability and Efficiency**)
- **RQ5**: How effective is FinDet in detecting adversarial contracts in real-world settings, and what is its potential impact in practical deployments? (**Real-World Impact**)

### B. RQ1: Baseline Comparison

To ensure fair comparison and reliable results, we use the same training and testing splits across all experiments. Specifically, we randomly select 20% of adversarial and benign contracts from  $\mathcal{D}_{GT}$  as the testing set, with the remaining 80% used for training. This process is repeated five times, and the average performance is reported. All baseline methods are trained and evaluated using these five data splits, except for the “unseen pattern” experiment in §VII-B2, which requires special consideration of adversarial contract types. As our method does not require training, we directly evaluate it on the five testing sets.

1) *Overall Performance*: TABLE II summarizes the detection performance of different methods. We observe that FinDet achieves a state-of-the-art balanced accuracy (BAC) of 0.9233, outperforming Lookahead (0.7446) and Skyeye (0.8814). The false negative rates (FNR) of Lookahead and Skyeye are 0.5100 and 0.2350, respectively, indicating their high miss rates in detecting adversarial contracts. In contrast, FinDet significantly reduces the FNR to 0.1050, demonstrating its superior capability in identifying attack behaviors.

<sup>1</sup><https://owasp.org/www-project-smart-contract-top-10/>

<sup>2</sup>[https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced\\_accuracy\\_score.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html)

TABLE III: Performance under the new type scenario (i.e., testing on unseen attack patterns). The terms “micro” and “macro” refer to methods of computing the final results after evaluating the model on 9 attack types and then aggregating the results.

Method	FPR ↓	FNR ↓	TPR ↑	TNR ↑	BAC ↑
Lookahead (micro)	0.0006	0.6610	0.3390	0.9994	0.6692
Lookahead (macro)	0.0004	0.7043	0.2957	0.9996	0.6476
Skyeye (micro)	0.0022	0.5085	0.4915	0.9978	0.7447
Skyeye (macro)	0.0028	0.4914	0.5086	0.9972	0.7529
FinDet	0.0484	<b>0.1050</b>	0.8950	0.9517	<b>0.9233</b>

TABLE II: Overall performance of different methods.

Method	FPR ↓	FNR ↓	TPR ↑	TNR ↑	BAC ↑
Lookahead	0.0008	0.5100	0.4900	0.9993	0.7446
Skyeye	0.0022	0.2350	0.7650	0.9979	0.8814
FinDet	0.0484	<b>0.1050</b>	0.8950	0.9517	<b>0.9233</b>

2) *Unseen Pattern Evaluation*: Our dataset includes adversarial contracts across nine categories. In each experiment, one category  $type_i$  is used as the test set. To ensure fairness, contracts labeled with both  $type_i$  and another category are excluded. If  $type_i$  has over 10 contracts, 10 are randomly selected for testing; otherwise, all are used. The training set is drawn from the remaining categories, capped at 100 contracts if available. This maintains a roughly 100:10 training-to-testing ratio. We report per-category results along with micro- and macro-averaged metrics.

TABLE III summarizes the results of baseline methods and FinDet under the new attack type evaluation. Lookahead’s FNR rises sharply to 0.6610 (micro) and 0.7043 (macro), reducing BAC from 0.7446 to 0.6692 and 0.6476, respectively. Skyeye shows a similar drop, with BAC decreasing from 0.8814 to 0.7447 (micro) and 0.7529 (macro), and FNR rising to over 0.49. In contrast, FinDet achieves the best BAC of 0.9233, with a low FNR of 0.1050, indicating strong generalization to unseen attack types. For example, when testing on Type 5 (reentrancy), Lookahead detects only 1 of 10 cases, and Skyeye detects 3, while FinDet performs significantly better.

3) *Training vs. Training-Free Performance*: To evaluate the impact of training set size on method performance, we conducted stratified sampling on the pre-partitioned training dataset, keeping the train-test split fixed for fair comparison.

TABLE IV presents results across various sampling ratios. Even at 10% (1,616 contracts), the training set remains larger than many datasets used by existing ML methods [39]. As training data decreases from 100% to 10%, Lookahead’s BAC drops from 0.7446 to 0.6918, Skyeye’s from 0.8814 to 0.8079, while FinDet consistently maintains a BAC of 0.9233, demonstrating robustness as a training-free method.

Notably, Lookahead and Skyeye require 40 and 12 hand-crafted features respectively, and the process of obtaining these features incurs **substantial extra cost** that is not included in the training time reported in TABLE IV. In contrast, FinDet only requires the contract bytecode as input, eliminating the need for any extra data preparation. The “Training Time” column in TABLE IV represents the total time required for model

training and inference after all necessary data preparation has been completed. Specifically, Lookahead requires up to 16.70 minutes, while Skyeye takes approximately 1,361.60 minutes (about 22 hours and 42 minutes). In comparison, FinDet is a training-free approach that requires no training time, making it significantly more efficient.

TABLE IV: Training vs. training-free performance. Training time is reported in minutes.

Method	Sample Ratio	FPR ↓	FNR ↓	TPR ↑	TNR ↑	BAC ↑	Training Time
Lookahead	100%	0.0008	0.5100	0.4900	0.9993	0.7446	16.70
	80%	0.0006	0.5150	0.4850	0.9995	0.7422	13.63
	60%	0.0008	0.5400	0.4600	0.9993	0.7296	10.45
	40%	0.0012	0.5450	0.4550	0.9988	0.7269	7.29
	20%	0.0017	0.6000	0.4000	0.9984	0.6992	4.10
	10%	0.0015	0.6150	0.3850	0.9985	0.6918	2.48
Skyeye	100%	0.0022	0.2350	0.7650	0.9979	0.8814	1361.60
	80%	0.0027	0.2550	0.7450	0.9974	0.8712	1245.59
	60%	0.0032	0.2600	0.7400	0.9969	0.8684	1068.70
	40%	0.0027	0.3250	0.6750	0.9973	0.8362	808.09
	20%	0.0038	0.3550	0.6450	0.9963	0.8206	458.99
	10%	0.0042	0.3800	0.6200	0.9958	0.8079	245.32
FinDet	<b>0%</b>	0.0484	<b>0.1050</b>	0.8950	0.9517	<b>0.9233</b>	<b>0.00</b>

4) *Obfuscation Evaluation*: Lookahead and Skyeye depend on on-chain transaction features like *Nonce*, *Value*, *InputDataLength*, *GasUsed*, and *Verified*, which are easily manipulated through simple obfuscation. Ma et al. [16] revealed that verified Ethereum contracts’ bytecode and source code may mismatch, exposing verification-breaking vulnerabilities with real exploitable cases. Inspired by this, we simulate an attacker submitting benign-appearing source code that doesn’t match deployed bytecode, setting the Verified feature to “True” for adversarial contracts to evade detection.

TABLE V shows that Lookahead and Skyeye’s FNRs increase (from 0.5100 to 0.5650 and from 0.2350 to 0.3250), and BACs drop (0.7170 and 0.8363). In contrast, FinDet, which doesn’t rely on on-chain features, remains unaffected. Other features like *Nonce* can also be obfuscated by sending benign transactions before deploying the real adversarial contract, disguising deployment patterns.

TABLE V: Performance under obfuscation scenarios.

Method	FPR ↓	FNR ↓	TPR ↑	TNR ↑	BAC ↑
Lookahead	0.0011	0.5650	0.4350	0.9990	0.7170
Skyeye	0.0025	0.3250	0.6750	0.9976	0.8363
FinDet	0.0484	<b>0.1050</b>	0.8950	0.9517	<b>0.9233</b>

**Summary to RQ1**: FinDet achieves a SOTA BAC of 0.9233 and low FNR of 0.1050, significantly better than baselines. FinDet remains robust under challenging scenarios where baseline performance degrades rapidly.

### C. RQ2: Impact of Behavioral Semantics

To evaluate the effectiveness of behavioral semantics, we introduce an additional baseline:

- **Baseline<sub>NL</sub>**: Natural language descriptions are fed directly into the LLM with a task-specific prompt (see Fig. 10).

We also conduct an ablation study by removing the general-purpose view (§IV-B, FinDet w/o G) and the attack-specific view (§IV-C, FinDet w/o S) to assess their individual contributions.

As shown in Table VI, removing the general-purpose view increases FPR to 0.2359, while removing the attack-specific view raises FNR to 0.2650. The corresponding BAC drops to 0.8371 and 0.8418, respectively. Baseline<sub>NL</sub> performs worst, with an FPR of 0.5668 and BAC of 0.7141. These results highlight the importance of both views for accurate detection.

TABLE VI: Performance comparison of different input information for Stage I. FinDet w/o G and FinDet w/o S refer to the ablation variants without the global-level contract view and the local-level behavior view, respectively.

Method	FPR ↓	FNR ↓	TPR ↑	TNR ↑	BAC ↑
FinDet w/o G	0.2359	0.0900	0.9100	0.7641	0.8371
FinDet w/o S	0.0515	0.2650	0.7350	0.9485	0.8418
Baseline <sub>NL</sub>	0.5668	0.0050	0.9950	0.4333	0.7141
FinDet	0.0484	0.1050	0.8950	0.9517	<b>0.9233</b>

#### D. RQ3: Impact of Confidence Probing and Fusion Strategy

In this section, we evaluate the effectiveness of our confidence probing and fusion strategy in Stage II. We compare our method with two alternatives: (1) FinDet w/o M, which removes the misleading probing prompts, and (2) Majority Voting, which aggregates predictions by voting based on the ranking order of adversarial and benign in each prompt. The results are summarized in TABLE VII. We observe that removing the misleading probing or replacing the entropy-based fusion strategy with a simpler voting mechanism leads to performance drops. This demonstrates that our method can better integrate predictions with varying confidence levels, resulting in improved overall detection performance.

TABLE VII: Performance comparison of different probing and fusion strategies. FinDet w/o M refers to the ablation variant without the misleading probing prompts.

Method	FPR ↓	FNR ↓	TPR ↑	TNR ↑	BAC ↑
FinDet w/o M	0.0903	0.0900	0.9100	0.9098	0.9099
Majority Voting	0.0406	0.2400	0.7600	0.9594	0.8597
FinDet	0.0484	0.1050	0.8950	0.9517	<b>0.9233</b>

#### E. RQ4: Scalability and Efficiency

1) *Evaluation under Different LLM Backbones:* To evaluate the generalizability of our approach, we conduct experiments using different LLM backbones, as shown in TABLE VIII. FinDet achieves consistently strong performance across models, with BAC scores of 0.8687 and 0.8319 on Claude 3.5 and GPT-4o, and correspondingly low FNRs of 0.0850 and 0.0150. Notably, the best performance is observed on DeepSeek-V3, with a BAC of 0.9233 and an FNR of 0.1050. These results demonstrate that FinDet maintains high detection accuracy across a range of LLMs, indicating good generalizability.

TABLE VIII: Performance under different LLM.

Model	FPR	FNR	TPR	TNR	BAC
Claude 3.5	0.1776	0.0850	0.9150	0.8224	0.8687
GPT-4o	0.3213	0.0150	0.9850	0.6787	0.8319
DeepSeek-V3	0.0484	0.1050	0.8950	0.9517	0.9233

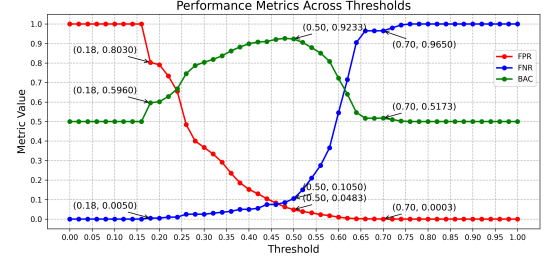


Fig. 9: Performance metrics across varying adversarial score thresholds.

2) *Optional Threshold Selection:* As discussed in §VI-D, users may have different preferences regarding the balance between FPR and FNR depending on specific application scenarios. To accommodate this, we provide an optional threshold selection mechanism, allowing users to adjust the adversarial score threshold based on their desired trade-off. Generally, a higher threshold yields lower FPR but higher FNR, while a lower threshold results in the opposite. As shown in Fig. 9, setting the threshold to 0.18 results in a very low FNR of 0.005, whereas increasing the threshold to 0.7 reduces the FPR to 0.0003. This illustrates the inherent trade-off between sensitivity and specificity in threshold-based decisions.

3) *Efficiency Analysis:* We compare average token and time costs across models to assess efficiency. For Deepseek-V3, the average token usage is 17,085.48, with a per-contract cost of \$0.0030: \$0.0018 (60.63%) from Stage I and \$0.0012 (39.37%) from Stage II. In comparison, Claude-3.5 costs \$0.0021 and GPT-4o \$0.0664 per contract. Since our method operates solely on bytecode, it supports pre-deployment detection and reduces overall overhead.

#### F. RQ5: Real-World Impact

To assess the real-world effectiveness of FinDet, we made our best effort to collect five publicly reported adversarial contracts (TABLE X) from reliable sources. FinDet successfully identified all of them based solely on bytecode, yielding a 100% true positive rate. Moreover, we randomly sampled contracts from  $\mathcal{D}_{RW}$  and, with expert-assisted manual validation, discovered 29 previously undocumented adversarial contracts exhibiting clear malicious intent or involvement in stealthy attacks.

1) *Case Study:* As part of our case study, we analyze an adversarial contract flagged by FinDet and independently confirmed by TenArmorAlert. The attack resulted in a loss of approximately \$26K. As the source code is unavailable, we integrate two state-of-the-art tools, Disco [31] for source code recovery and Elipmoc [9] for pseudocode lifting, together with manual validation to ensure that the recovered logic is



semantically consistent with the on-chain runtime bytecode, as shown in Fig. 4.

As analyzed in **Observation 2**, this contract executes its attack in typical three stages: preparation (injecting funds via `exploit()`), exploitation (arbitrage via `DPPFlashLoanCall()`), and exit (extracting illicit gains in `DPPFlashLoanCall()`). FinDet extracted key fund-flow paths (Fig. 15). TABLE XII shows prediction scores from 6 prompts. After entropy-based fusion, the adversarial, suspicion, uncertain, and benign scores are 0.288, 0.327, 0.221, and 0.163, respectively. The combined adversarial score (adversarial + suspicion) is 0.615, leading to a final prediction of adversarial.

2) *Real World Findings*: Notably, FinDet identified instances of price manipulation and arbitrage facilitated by flash loans, as illustrated in Fig. 4. These cases represent a rapidly emerging and increasingly prevalent threat in recent years, distinctly different from conventional flash loan attacks that exploit protocol vulnerabilities, as well as from benign arbitrage strategies that leverage price inefficiencies without adversarial manipulation. The detected behaviors involve adversaries borrowing large amounts of assets via flash loans to actively distort price dynamics across protocols and extract profits through carefully orchestrated arbitrage. These cases conform to the three-stage adversarial pattern abstracted in our model, highlighting FinDet’s capability to capture emerging yet previously unseen attack patterns.

## VIII. DISCUSSION

**Limitations.** Our method detects adversarial behavior exhibited directly by contracts. It may miss proxy contracts that delegate malicious logic elsewhere, as these proxies show no adversarial actions themselves. For example, some false negatives occur when the actual exploit lies in a downstream callee, not the proxy.

**Future Work.** Currently, we analyze adversarial contracts independently, focusing on internal logic and fund flows. However, many attacks involve subtle interactions with victim contracts. Future work could integrate victim contracts into the analysis, enabling joint reasoning over attacker and target behaviors. Since most adversarial contracts hardcode victim addresses or store them in fixed slots [43], cross-contract analysis is both feasible and promising. This would improve detection of coordinated attacks (e.g., reentrancy, storage collisions), track fund flows across contracts, and enhance accuracy in complex scenarios.

## IX. RELATED WORK

### A. Vulnerability Detection in Smart Contracts

Vulnerability detection is a long-standing and critical task in the blockchain ecosystem. Traditional methods can be broadly categorized into four types: static analysis-based [8, 10], symbolic execution-based [36], fuzzing-based [7, 35], and machine learning-based approaches [20, 46]. LLMs have recently been explored for vulnerability detection [13], with hybrid methods combining them with static analysis [33], multi-agent reasoning [17], or fuzzing [28].

Despite recent progress, existing approaches face three key limitations. First, most detected vulnerabilities are not practically exploitable: only 2.68% [43] of flagged contracts are actually exploited. Second, high false positives remain common [30], such as misclassifying safe anti-reentrancy contracts as vulnerable. Third, there remains an inherent asymmetry: defenders must address all threats, while attackers only need to exploit one. These issues highlight the need for proactive, adversary-aware detection focused on real-world exploits rather than theoretical flaws.

### B. Adversarial Detection in Transactions

Some works aim to detect attacks in real time by monitoring unconfirmed transactions in the public mempool. They typically follow two approaches: pattern-based [37, 40, 41], using manually defined heuristics; and learning-based [4, 12, 23], leveraging ML models with graph-based features.

While effective in public settings, these methods fail to detect transactions submitted via private mempools, which bypass public monitoring. Moreover, their reliance on fixed rules or models limits adaptability to novel or evolving attacks.

### C. Adversarial Detection in Smart Contracts

Beyond vulnerability identification, recent research has focused on detecting adversarial smart contracts deliberately designed to exploit vulnerabilities or manipulate protocol logic. For example, Yang et al. [43] propose a method to identify contracts capable of launching exploitable reentrancy attacks. SMARTCAT [44] proposes an efficient static analyzer that detects price manipulation adversarial contracts via cross-function call graphs and token flow graphs. However, these approaches target only a specific type of adversarial contracts.

In contrast to pattern-specific detection, two works aim to develop ML-based systems that generalize across various types of adversarial contracts. Lookahead [24] lifts bytecode into three-address code using Elipmoc [9], and combines Pruned Semantic-Control Flow Tokenization with handcrafted features using a dual-branch architecture for final classification. Skyeeye [38] extracts static features and performs control flow graph segmentation to build a unified representation for training a binary classifier. However, ML-based approaches heavily rely on labeled training data, limiting their ability to detect novel or unseen attack variants. They are trained on low-level opcode features, lacking the capacity to capture high-level semantic intent. Furthermore, their poor interpretability hinders security analysts from understanding model decisions, which is critical in high-stakes security contexts.

## X. CONCLUSION

In this work, we presented FinDet, a novel and training-free framework for detecting adversarial smart contracts directly from EVM bytecode. By lifting low-level bytecode into semi-structured natural language, FinDet enhances semantic understanding and enables multi-perspective analysis of contract behavior. Additionally, FinDet incorporates fund-flow reachability analysis to capture the distinct stages of attacks, thereby strengthening semantic precision. To further improve robustness, we introduced a fine-grained uncertainty quantification mechanism that mitigates hallucinations and

enhances detection reliability. Our method demonstrates strong generalization to unseen attack patterns, resilience in low-data settings, and state-of-the-art performance across multiple metrics. These results highlight the potential of LLM-based semantic reasoning for proactive and effective smart contract security. We believe FinDet offers a practical step toward securing blockchain ecosystems against emerging adversarial threats.

## REFERENCES

- [1] A. Amin, S. Anwar, A. Adnan, M. Nawaz, N. Howard, J. Qadir, A. Hawalah, and A. Hussain, "Comparing oversampling techniques to handle the class imbalance problem: A customer churn prediction case study," *Ieee Access*, vol. 4, pp. 7940–7957, 2016.
- [2] Anthropic, "Claude: Haiku," <https://www.anthropic.com/claude/haiku>, 2023, accessed: 2025-07-30.
- [3] P. C. Aversaccio, "Reentrancy attacks github repository," <https://github.com/pcaversaccio/reentrancy-attacks>, 2025, accessed: 2025-06-26.
- [4] K. Babel, M. Javaheripi, Y. Ji, M. Kelkar, F. Koushanfar, and A. Juels, "Lanturn: Measuring economic security of smart contracts through adaptive learning," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 1212–1226.
- [5] Breadcrumbs App. (2021) Thorchain heist: An analysis of the fund flow. Accessed: 2025-07-09. [Online]. Available: <https://medium.com/breadcrumbsapp/thorchain-heist-an-analysis-of-the-fund-flow-7ca18a18d044>
- [6] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, "The balanced accuracy and its posterior distribution," in *2010 20th international conference on pattern recognition*. IEEE, 2010, pp. 3121–3124.
- [7] J. Choi, D. Kim, S. Kim, G. Grieco, A. Groce, and S. K. Cha, "Smartian: Enhancing smart contract fuzzing with static and dynamic data-flow analyses," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 227–239.
- [8] J. Feist, G. Grieco, and A. Groce, "Slither: a static analysis framework for smart contracts," in *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. IEEE, 2019, pp. 8–15.
- [9] 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.
- [10] I. Grishchenko, M. Maffei, and C. Schneidewind, "Ethertrust: Sound static analysis of ethereum bytecode," *Technische Universität Wien, Tech. Rep.*, pp. 1–41, 2018.
- [11] S. Hu, T. Huang, K.-H. Chow, W. Wei, Y. Wu, and L. Liu, "Zipzap: Efficient training of language models for large-scale fraud detection on blockchain," in *Proceedings of the ACM Web Conference 2024*, 2024, pp. 2807–2816.
- [12] Z. Li, J. Li, Z. He, X. Luo, T. Wang, X. Ni, W. Yang, X. Chen, and T. Chen, "Demystifying defi mev activities in flashbots bundle," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 165–179.
- [13] J. Lin and D. Mohaisen, "From large to mammoth: A comparative evaluation of large language models in vulnerability detection," in *Proceedings of the 2025 Network and Distributed System Security Symposium (NDSS)*, 2025.
- [14] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.
- [15] 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," *arXiv preprint arXiv:2405.02580*, 2024.
- [16] P. Ma, N. He, Y. Huang, H. Wang, and X. Luo, "Abusing the ethereum smart contract verification services for fun and profit," *arXiv preprint arXiv:2307.00549*, 2023.
- [17] 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," *arXiv preprint arXiv:2403.16073*, 2024.
- [18] Merkle Science. (2023) Onyx protocol hack: Flow of funds analysis. Accessed: 2025-07-09. [Online]. Available: <https://www.merklescience.com/blog/onyx-protocol-hack-flow-of-funds-analysis>
- [19] L. Mosley, "A balanced approach to the multi-class imbalance problem doctoral dissertation," *Iowa State University of Science and Technology, USA*, 2013.
- [20] H. H. Nguyen, N.-M. Nguyen, H.-P. Doan, Z. Ahmadi, T.-N. Doan, and L. Jiang, "Mando-guru: Vulnerability detection for smart contract source code by heterogeneous graph embeddings," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 1736–1740.
- [21] OpenAI, "GPT-4o Model," <https://platform.openai.com/docs/models/gpt-4o>, accessed: 2025-06-30.
- [22] D. Perez and B. Livshits, "Smart contract vulnerabilities: Vulnerable does not imply exploited," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1325–1341.
- [23] P. Qian, J. He, L. Lu, S. Wu, Z. Lu, L. Wu, Y. Zhou, and Q. He, "Demystifying random number in ethereum smart contract: Taxonomy, vulnerability identification, and attack detection," *IEEE Transactions on Software Engineering*, vol. 49, no. 7, pp. 3793–3810, 2023.
- [24] S. Ren, L. He, T. Tu, D. Wu, J. Liu, K. Ren, and C. Chen, "Lookahead: Preventing defi attacks via unveiling adversarial contracts," *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, pp. 1847–1869, 2025.
- [25] M. Science. (2024) Hack track: Deltaprime flow of funds analysis. Accessed: 2025-07-09. [Online]. Available: <https://www.merklescience.com/blog/hack-track-deltaprime-flow-of-funds-analysis>
- [26] C. Sendner, L. Petzi, J. Stang, and A. Dmitrienko, "Large-scale study of vulnerability scanners for ethereum smart contracts," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 2273–2290.
- [27] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.
- [28] C. Shou, J. Liu, D. Lu, and K. Sen, "Llm4fuzz: Guided fuzzing of smart contracts with large language models," *arXiv preprint arXiv:2401.11108*, 2024.
- [29] SlowMist, "Slowmist hacked statistics 2024," <https://hacked.slowmist.io/statistics/?c=all&d=2024>, SlowMist, 2024, accessed: 2025-07-04.
- [30] Q. Song, H. Huang, X. Jia, Y. Xie, and J. Cao, "Silence false alarms: Identifying anti-reentrancy patterns on ethereum to refine smart contract reentrancy detection," in *The Network and Distributed System Security (NDSS) Symposium 2025*, 2025.
- [31] X. Su, H. Liang, H. Wu, B. Niu, F. Xu, and S. Zhong, "Disco: Towards decompiling evm bytecode to source code using large language models," *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, pp. 2311–2334, 2025.
- [32] J. Sun, Z. Yin, H. Zhang, X. Chen, and W. Zheng, "Adversarial generation method for smart contract fuzz testing seeds guided by chain-based llm," *Automated Software Engineering*, vol. 32, no. 1, pp. 1–28, 2025.
- [33] 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.
- [34] SunWeb3Sec, “DeFiHackLabs github repository,” <https://github.com/SunWeb3Sec/DeFiHackLabs>, 2025, accessed: 2025-06-26.
- [35] C. F. Torres, A. K. Iannillo, A. Gervais, and R. State, “Con-fuzzius: A data dependency-aware hybrid fuzzer for smart contracts,” in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2021, pp. 103–119.
- [36] A. Wang, H. Wang, B. Jiang, and W. K. Chan, “Artemis: An improved smart contract verification tool for vulnerability detection,” in *2020 7th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE, 2020, pp. 173–181.
- [37] D. Wang, S. Wu, Z. Lin, L. Wu, X. Yuan, Y. Zhou, H. Wang, and K. Ren, “Towards a first step to understand flash loan and its applications in defi ecosystem,” in *Proceedings of the Ninth International Workshop on Security in Blockchain and Cloud Computing*, 2021, pp. 23–28.
- [38] H. Wang, Y. Hu, H. Wu, D. Liu, C. Peng, Y. Wu, M. Fan, and T. Liu, “Skyeye: Detecting imminent attacks via analyzing adversarial smart contracts,” in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 1570–1582.
- [39] C. Wu, J. Chen, Z. Wang, R. Liang, and R. Du, “Semantic sleuth: Identifying ponzi contracts via large language models,” in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 582–593.
- [40] 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*, vol. 21, no. 4, pp. 4147–4161, 2023.
- [41] 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.
- [42] M. Xiong, Z. Hu, X. Lu, Y. Li, J. Fu, J. He, and B. Hooi, “Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms,” in *ICLR*, 2024.
- [43] S. Yang, J. Chen, M. Huang, Z. Zheng, and Y. Huang, “Uncover the premeditated attacks: Detecting exploitable reentrancy vulnerabilities by identifying attacker contracts,” in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–12.
- [44] B. Zhang, N. He, X. Hu, K. Ma, and H. Wang, “Following devils’ footprint: Towards real-time detection of price manipulation attacks,” *arXiv preprint arXiv:2502.03718*, 2025.
- [45] H. Zhao, H. Chen, F. Yang, N. Liu, H. Deng, H. Cai, S. Wang, D. Yin, and M. Du, “Explainability for large language models: A survey,” *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 2, pp. 1–38, 2024.
- [46] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, “Smart contract vulnerability detection using graph neural networks,” in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2021, pp. 3283–3290.

## APPENDIX

### A. Dataset Detail

Table IX shows the detailed category distribution of our dataset  $\mathcal{D}_{GT}$ .

<sup>3</sup>Some contracts involve multiple attack types, so the total count may not match the sum of all categories. No DoS-related contracts were observed during data collection.

TABLE IX: Different Adversarial Contract Types in  $\mathcal{D}_{GT}$

Type ID	Attack Type	Contract Number
Type1	Access Control Vulnerabilities	13
Type2	Price Oracle Manipulation	109
Type3	Logic Errors	29
Type4	Lack of Input Validation	7
Type5	Reentrancy Attacks	53
Type6	Unchecked External Calls	2
Type7	Flash Loan Attacks	21
Type8	Integer Overflow and Underflow	3
Type9	Insecure Randomness	5
Type10	Denial of Service (DoS) Attacks	0
<b>Total</b>		<b>200<sup>3</sup></b>

### Prompt Template of Baseline<sub>NL</sub>

{Natural language description of smart contract.}

This text provides a natural language description of a smart contract deployed on the blockchain. Based on this description, determine whether the contract is adversarial or benign. Please respond using the following format:

1) [[adversarial/benign]]

2) [[Your reasoning for this judgment]]

Fig. 10: Prompt template for Baseline<sub>NL</sub>.

### B. Prompt of Baseline<sub>NL</sub>

Fig. 10 shows the prompt template of Baseline<sub>NL</sub>.

### C. Disclosed Attack Cases

Table X lists adversarial contracts detected by FinDet that have been publicly disclosed.

TABLE X: Real-world attack cases detected by FinDet and publicly disclosed.

Deploy Time	Report	Adversarial Contract Address
2025.04.18	link	0x7a4d144307d2dfa2885887368e4cd4678db3c27a
2025.04.23	link	0xdd9a85fd532faadb0c439bbd725e571c4214aedef
2025.04.26	link	0xf6cee497dfe95a04faa26f3138f9244a4d92f942
2025.04.26	link	0x75f2002937507b826b727170728595fd45151d0f
2025.04.26	link	0xcfd3cf61619cbe15e9a8bef0e5cd613a565b6b3

### D. Supplementary Prompts in Stage II

Fig. 11 and Fig. 12 shows the prompts used for general-purpose analysis and attack-specific analysis in Stage II.

### E. Motivation Example: Analysis of FinDet

In this section, we provide a detailed analysis of FinDet, referring to the motivation example shown in Fig. 4 and the case study (§VII-F1) presented earlier.

### F. Compared with Baseline<sub>NL</sub>: A benign example

We conduct a detailed comparison using the contract at <https://etherscan.io/address/0x5c1bfab1f6bbd8f059c7d0c0124e5b8a7bff84fd> as a representative case study. This example supports both Observation 2 and Observation 3 in §III-B.

## Prompt Template of General-Purpose Analysis

### === Contract-Level Information ===

(This section was generated by this LLM in a previous stage, based on the natural language description of the decompilation of this smart contract)

Contract Summary: {Contract summary from stage I}

### Static Structural Analysis

- **External Call Behavior**  
External calls often indicate reliance on external components, which is a common feature in adversarial contracts (e.g., flash loans, oracle manipulation, reentrancy). High external call ratio may indicate increased complexity or obfuscation. Total external calls: {External call number}, External call ratio: {External call ratio}
- **Unknown Function Analysis**  
Adversarial contracts often use unnamed, obfuscated, or auto-generated functions to hide malicious logic. These functions lack clear semantics and often bypass detection. Unknown function count: {Unknown function number}, Unknown function ratio: {Unknown function ratio}
- **Fund Transfer Behavior**  
In adversarial contracts, fund transfers are often essential for extracting assets, whereas benign contracts typically implement such transfers with clear authorization and legitimate purposes. Transfers in unknown functions: {Yes/No}
- **Bot-related Functions**  
Bot-related functions such as addBot or delBot are common in benign contracts but rarely appear in adversarial contracts, which tend to minimize logic and avoid non-essential features. Bot function count: {Bot function number}, Bot function ratio: {Bot function ratio}
- **Clarification on Common Patterns in Token Contracts**
  - Minting operations that assign tokens from the zero address (e.g., `from == 0x0`) are standard in ERC721 and similar token standards. This should not be misinterpreted as a zero-address exploit.
  - The presence of `delegatecall` is often part of proxy or upgradeable contract designs and should not be flagged as adversarial without analyzing the surrounding structure and intent.
  - Complex conditional logic or the use of seemingly arbitrary constants (e.g., in transfer or anti-bot logic) may serve legitimate purposes and are not inherently malicious.
  - Common token patterns such as allowance adjustments, fee-on-transfer, or mint/burn functions are benign unless clear misuse or deviation is detected.
  - The existence of privileged operations or ownership controls is normal in many contracts. Attention should be placed on whether such privileges are abused, lack access controls, or serve malicious intent—rather than their mere presence.

Fig. 11: The prompt for general-purpose analysis in Stage II.

## Prompt Template of Attack-specific Analysis

### === Function-Level Information ===

(This section was generated by this LLM in a previous stage, based on the natural language description of the decompilation of this smart contract)

- (i-th function) {Function name}: {Function purpose}, Suspicious: {Suspicious}, Reason: {Reason}

### Unknown Function Descriptions

(This section lists all functions with names starting with “unknown”, typically indicating unidentified or obfuscated logic)

- (i-th function) {Function name}: {Function parameters}, Description: {NL description}, Reason: {Reason}

### === Fund Flow-Level Information ===

(The following paths are generated by fund-flow reachability analysis)

Fund-Flow Paths (ingress to egress):

Format: `ingress_var -[cond1, cond2]-> var2 -[cond3]-> egress_var`  
{Path 1: fund-flow path 1}, {Path 2: fund-flow path 2}, ...

Fig. 12: The prompt for attack-specific analysis in Stage II.

The results for Baseline<sub>NL</sub> reveal a consistent tendency to focus on low-level suspicious patterns rather than understanding the high-level semantic intent of the contract be-

### • DPPFlashLoanCall:

- purpose: This function initiates a series of external calls to approve, transfer, buy, and sell tokens, with specific conditions and checks on return data.
- suspicious: Yes.
- reason: The function requires a specific hash of the transaction origin, which is highly unusual and suggests it may be designed to only allow execution by a specific entity, potentially indicating a backdoor or exploit.

### • receiveFlashLoan:

- purpose: This function handles the receipt of a flash loan, transferring amounts and checking balances.
- suspicious: Yes.
- reason: Similar to the previous function, the requirement for a specific hash of the transaction origin suggests restricted access, which could be used to exploit the contract.

### • unknownfffcf3a1:

- purpose: This function checks a balance and initiates a flash loan with specific parameters.
- suspicious: Yes.
- reason: The requirement for a specific hash of the transaction origin indicates potential malicious intent, as it restricts function execution to a specific entity.

Fig. 13: Function-centric summary of the contract in Fig. 4

- **contract summary:** The contract appears to be designed to perform complex financial operations involving flash loans, token transfers, and balance checks. However, it includes specific restrictions on transaction origin, suggesting it may be intended for use by a single entity, potentially for exploitative purposes.

Fig. 14: Contract-centric summary of the contract in Fig. 4

havior, which results in overly conservative judgments and a high rate of false positives. For instance, in Observation 2, Baseline<sub>NL</sub> labels a contract as adversarial primarily because it “contains several unusual patterns and complex logic that are often seen in adversarial contracts designed to exploit or bypass security measures,” and “performs complex and numerous checks on member access, transaction amounts, and wallet sizes for transfers.” However, these behaviors are often part of legitimate access-control and compliance logic in production contracts. This illustrates a form of “loss of contextual focus” where the model misinterprets defensive or regulatory mechanisms as adversarial strategies. Furthermore, as shown in Observation 3, Baseline<sub>NL</sub> tends to default to adversarial predictions in ambiguous gray-area scenarios, such

Prompt	G1	P1	G2	P2	G3	P3	G4	P4
prompt_g_normal	B	60	A	25	C	10	D	5
prompt_s_normal	B	60	A	30	C	8	D	2
prompt_g_mislead_ad	A	60	B	30	C	8	D	2
prompt_g_mislead_be	D	70	C	20	B	8	A	2
prompt_s_mislead_ad	A	80	B	15	C	5	D	0
prompt_s_mislead_be	D	60	C	30	B	10	A	0

TABLE XI: Predictions across different prompt settings by FinDet for the contract in Fig. 4.



**Path 1:**  
 caller -[it is required that (0x268d...4080 == sha3(tx.origin)), it is required that the 2nd external call succeeds, it is required that the 3rd external call succeeds, When (param1 > 0), it is required that the 5th external call succeeds, it is required that the 1st external call succeeds, it is required that ((90 \* the 1st return data of the 5th external call) / 100), it is required that the 1st return data of the 3rd external call]-> stor\_3.transfer

**Path 2:**  
 DPPFlashLoanCall:param1 -[it is required that (0x268d...4080 == sha3(tx.origin)), it is required that the 2nd external call succeeds, it is required that the 3rd external call succeeds, When (param1 > 0), it is required that the 5th external call succeeds, it is required that the 1st external call succeeds, it is required that ((90 \* the 1st return data of the 5th external call) / 100), it is required that the 1st return data of the 3rd external call]-> stor\_3.transfer

**Path 3:**  
 unknownfffcf3al:param1 -[it is required that (0x268d...4080 == sha3(tx.origin)), it is required that the 1st external call succeeds]-> stor\_5.flashLoan

Fig. 15: Fund-Flow analysis of the contract in Fig. 4.

as contracts interacting with external components in non-standard ways, due to a lack of nuanced reasoning. Together, these behaviors point to the model’s over-reliance on surface-level suspicious features without integrating broader semantic context, leading to misclassification of benign contracts.

In contrast, FinDet employs a multi-view, high-level behavioral semantic analysis. This comprehensive and multi-dimensional approach enables FinDet to differentiate among “suspicious,” “uncertain,” and “benign” signals effectively. The scores from the six prompts are presented in Table X. After applying entropy-based fusion, the aggregated scores for *adversarial*, *suspicion*, *uncertain*, and *benign* are 0.1211, 0.2472, 0.3225, and 0.3092, respectively. The combined adversarial score (adversarial + suspicion) is 0.37, which leads to the successful classification of the contract as benign.

In summary, our findings confirm that Baseline<sub>NL</sub>’s isolated focus on suspicious indicators leads to overly conservative and erroneous adversarial predictions. In contrast, FinDet’s holistic analysis framework produces more accurate and interpretable judgments, thereby demonstrating the importance of multi-level reasoning and confidence-based probing and fusion for robust adversarial contract detection.

Prompt	G1	P1	G2	P2	G3	P3	G4	P4
prompt_g_normal	D	50	C	30	B	15	A	5
prompt_s_normal	B	50	C	30	A	15	D	5
prompt_g_mislead_ad	C	40	D	30	B	20	A	10
prompt_g_mislead_be	D	70	C	20	B	8	A	2
prompt_s_mislead_ad	A	70	B	20	C	8	D	2
prompt_s_mislead_be	D	70	C	20	B	8	A	2

TABLE XII: Predictions across different prompt settings by FinDet for the contract in Appendix F.

#### Analysis of Baseline<sub>NL</sub>:

- The contract contains several unusual patterns and complex logic that are often seen in adversarial contracts designed to exploit or bypass security measures.
- There are numerous checks and updates of state variables that appear convoluted and overlapping, which might be intended to obfuscate the true purpose of the contract.
- There are repetitive and redundant operations around balance adjustments and transfer mechanisms, including states like 'stor\_19\_21\_22', whose intended roles and necessity are not clear, raising concerns about hidden or malicious functionality.
- The contract involves frequent interaction with an external contract, as seen from multiple external calls, which could be exploited if not properly managed or secured.
- Lack of clear and straightforward logic, with many nested conditions and branches, often makes it easier for an adversary to insert malicious logic without immediate detection.
- The presence of operations that transfer zero wei or call external functions with seemingly unnecessary actions could be a technique to cloak malicious payloads or test system constraints.
- The contract performs complex and numerous checks on member access, transaction amounts, and wallet sizes for transfers, which could be indicative of attempts to deliberately manipulate such transactions for unauthorized gains.
- The contract utilizes low-level operations ('ext\_code.size', external calls) with conditions that may attempt to manipulate contract behavior unexpectedly, showing suspicious activity aligned with adversarial intent.

#### Final Prediction:

Adversarial



Fig. 16: Baseline<sub>NL</sub> detection results for a benign contract in Appendix F.