# GenDetect: Generalizing Reactive Detection for Resilience Against Imitative DeFi Attack Cascade

Bowen Cai
cai00254@umn.edu
University of Minnesota
Minneapolis, MN, USA

Weiheng Bai
bai00093@umn.edu
University of Minnesota
Minneapolis, MN, USA

Youshui Lu
yolu6176@uni.sydney.edu.au
Xi'an Jiaotong University
Xi'an, Shaanxi, China

Haoran Xu
hxu65@jh.edu
Johns Hopkins University
Baltimore, Maryland, USA

Yuannan Yang
yyang181@jh.edu
Johns Hopkins University
Baltimore, Maryland, USA

Yajin Zhou
yajin_zhou@zju.edu.cn
Zhejiang University
Hangzhou, Zhejiang
China

Kangjie Lu
kjlu@umn.edu
University of Minnesota
Minneapolis, MN, USA

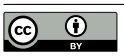
## Abstract

As blockchain ecosystems grow, financially motivated attackers have increasingly exploited vulnerabilities in decentralized finance (DeFi) protocols, resulting in frequent and severe losses. Unlike conventional cyberattacks, DeFi exploits propagate rapidly due to the transparent and composable nature of smart contracts. In this setting, we identify a critical behavioral pattern: *Imitative Attack Cascade*, where an initial successful exploit is quickly followed by a flurry of mimicking transactions that reuse attack logic with minor modifications or parameter changes. Our empirical analysis shows that over 69% of DeFi attacks exhibit strong behavioral similarity to earlier incidents, often occurring within hours or days of the initial attack.

This phenomenon highlights a fundamental limitation in current reactive detection workflows. While the initial attacks are often flagged through heuristic alerts, such as Tornado Cash traces, anomalous nonce usage, or known exploiter labels, these signals require manual validation and the construction of handcrafted detection rules through trace analysis. This process is labor-intensive and slow, resulting in unacceptable latency while follow-up attacks continue to spread. Motivated by this gap, our research goal is to ensure that once an attack has been observed, even a single instance, it can be rapidly abstracted into an actionable and generalizable detection rule, enabling scalable protection against imitative attacks.

We decompose the problem into two core challenges: (I) abstracting the semantics of diverse, obscure function signatures, and (II) matching transaction logic in noisy, evasive traces. To address these, we leverage two key insights: (i) the open-source nature of most DeFi protocols enables high-fidelity semantic classification of function signatures; (ii) contract labels allow us to isolate essential logic by filtering irrelevant calls and classifying attack intent. Based on these, we develop a reactive detection framework, *GenDetect*, which achieves strong benchmark performance (ACC: 98%, FPR: 1%, FNR: 3%) and, critically, discovers 56 previously unrevealed attacks from the past three years, highlighting its practical effectiveness. Our source code and dataset are released on Github.

**ACM Reference Format:**
Bowen Cai, Weiheng Bai, Youshui Lu, Haoran Xu, Yuannan Yang, Yajin Zhou, and Kangjie Lu. 2026. GenDetect: Generalizing Reactive Detection for Resilience Against Imitative DeFi Attack Cascade. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3744916.3787799




*ICSE '26, Rio de Janeiro, Brazil*

ACM ISBN 979-8-4007-2025-3/2026/04
https://doi.org/10.1145/3744916.3787799

## 1 Introduction

The rapid growth of blockchain ecosystems has been accompanied by a sharp rise in security threats. Each year, hundreds of security incidents result in billions of dollars in losses [3]. Among these, Decentralized Finance (DeFi) [13, 26, 32, 49] stands out as the most prominent and lucrative target due to its high-value assets and composable smart contract design. The transparency and reusability of smart contracts make successful exploits particularly easy to replicate, contributing to the rapid proliferation of follow-up attacks. Yet detection methods remain inadequate in the face of these escalating threats. Industry solutions [37] often rely on ad hoc heuristics–such as hardcoded exploiter addresses, Tornado Cash interactions [33], or timing-based indicators like nonce or contract creation time–that may help identify the first exploit but fail to capture its evolving variants. Academic approaches, on the other hand, typically focus on specific vulnerability types (e.g., price manipulation [31, 43, 44, 46], flashloans [47], or reentrancy [51]), lacking the generality required to address real-world exploits at scale. As a result, once a vulnerability is exposed, it is often repeatedly exploited–sometimes for months–due to delays in designing and deploying effective detection patterns.

**The Imitative DeFi Attack Cascade.** A key motivation for our research lies in the empirical observations that most DeFi attacks are not isolated or fundamentally novel, but rather imitations, adaptations, or direct replications of previously disclosed exploits. In real-world incidents, once an attack is proven effective, it often triggers an *imitative attack cascade*–sometimes reusing entire contracts with minimal modifications, and sometimes varying parameters or call sequences. To quantify this pattern, we analyze confirmed attack transactions from Phalcon [18] and DeFiHackLab [42] across

multiple protocols. Our analysis reveals that over 69% of the attacks–collectively accounting for more than $2 billion in losses–exhibit strong behavioral similarities to the earlier exploits. Some follow-ups occur within hours or days of the original, while others replicate nearly identical strategies over a year later. This persistence underscores not only the reusability of vulnerable patterns but also the absence of timely and generalizable defenses. Without a mechanism to convert each new exploit into a broadly applicable detection pattern, today's undetected attack becomes tomorrow's widely reused blueprint. Detection generalization is therefore not a supplementary feature–it is the core capability needed for sustainable defense.

**Prior Work.** In practice, the industry relies on broad, heuristic-based signals–such as low-nonce initiators, Tornado Cash funding paths, or known exploiter addresses–to raise alerts [19]. Although these methods offer wide coverage and increase the odds of catching the first exploit, they suffer from extremely high false positives and require substantial manual effort to verify. More importantly, even when the first attack is confirmed, such approaches cannot automatically generate rules to detect follow-up *imitative attacks*. This limitation stems from the fact that most existing detection systems follow a reactive loop of "discovery→manual analysis→rule generation", as seen in TxSpector [51], DeFiRanger [44], and POMA-Buster [46]. These tools manually extract behavioral signatures from traces to produce handcrafted rules, but their reliance on human intervention introduces unacceptable latency–especially in blockchain environments where imitation attacks can emerge within minutes. Alarmingly, certain vectors remain exploitable months or even years after disclosure [14–17]–not due to ignorance, but because current defense systems lack the capability to generalize and automate detection across related contexts.

This highlights the core value of our work: Rather than aiming to be the first to discover new attacks, we take the initial disclosure–identified through external alerts, public reports, or monitoring tools–as input. By capturing such cases through **generalizable semantic representations**, we can **rapidly and automatically produce detection strategies**, avoiding the delays and brittleness of handcrafted rules. While no system can anticipate entirely new categories in advance, our method ensures that once an incident is revealed, any imitative form of attack is unlikely to happen stealthily. In this way, we complement existing efforts by bridging the gap between discovery and timely, scalable detection for *imitative attacks*.

**Research Goal and Challenges.** In this paper, our goal is to rapidly generalize an initial attack incident into a reactive detection to enable the precise resilience for *imitative DeFi attacks*. We decompose the goal into two main challenges: (C1) Semantic Abstraction of Function Signatures. Transaction traces often contain diverse and complex function signatures. Existing word-level approaches, such as keyword matching or word-vector embeddings, struggle to capture their true semantics, especially in the context of DeFi-specific operations. (C2) Robust Matching of Transaction Logic. DeFi transaction traces are typically noisy, containing numerous benign operations or even intentionally obfuscated behaviors. This severely undermines the effectiveness of naive pattern-matching techniques like Longest Common Subsequence (LCS), which are sensitive to trace variations and cannot tolerate structural disguises.

We leverage two key insights to address the challenges respectively: (I) Source-based Semantic Extraction. We observe that most real-world attacks primarily target valuable DeFi protocols, which are typically open-sourced. This allows us to go beyond surface-level function names and directly utilize source code to classify function signatures based on their actual implementations. Compared to word-level recognition, our method yields more semantically accurate representations of function behavior. (II) Contract label-based Logic Matching. We observe that a transaction's core intent often emerges from direct invocations by Externally Owned Accounts (EOAs) or affiliated contracts, particularly when these calls interact with two categories of targets: ① Core asset contracts (e.g., ETH, WBNB, USDT) for value extraction, and ② Protocol-specific tokens for exploitation. By isolating such initiators and grouping their interactions by target type, we reduce semantic noise and improve the robustness of logic matching against noisy even evasive behaviors.

**Evaluation Summary.** We evaluate our system on both benchmark and real-world datasets, demonstrating strong performance across accuracy, generalizability, and real-time capability. On the DeFiHackLab [42] benchmark, our approach achieves a cross-validation F1-score of 98%, with a false positive rate under 1%–substantially lower than the 16% reported by the state-of-the-art tool Forta [24]. Our ablation study further confirms the effectiveness of our design: Insight I contributes a 50% relative improvement, and Insight II adds 9%. In zero-shot evaluation on the Phalcon [18] dataset, our method achieves 76% accuracy, compared to 65% by Forta. Beyond known benchmarks, our system identifies 56 (§5.2) previously undisclosed malicious incidents involving over $1.5M in value. In terms of efficiency, our system supports transaction-level parallelism and meets the real-time requirements of most mainstream blockchains.

**Contributions.** We make the following key contributions:

- We present the first reactive detection generalization system that translates a single observed exploit into reusable detection for future imitative attacks.
- We introduce a semantic extraction and logic matching framework that mitigates false positives caused by obfuscated benign transactions and improves resilience against evasive attack patterns, enabling high-fidelity detection.
- We implement a high-speed, high-precision detection pipeline that supports real-time analysis, meeting practical latency requirements for on-chain monitoring.
- Our system successfully uncovers 56 previously unrevealed attacks in real-world transaction datasets, demonstrating its practical utility and coverage beyond known exploits.

## 2 Background and Motivation

### 2.1 Blockchain and DeFi

Blockchain is a decentralized ledger that allows immutable and transparent record-keeping across a distributed network. Among its most influential innovations is the smart contract, a self-executing program deployed on the blockchain that enables programmable and automated interactions. Prominent platforms such as Ethereum [21] and Solana [40] provide robust infrastructures for smart contract execution, which serve as a foundational layer for

decentralized financial applications (DeFi). These contracts operate autonomously once deployed, managing assets and logic without further intervention–making them powerful yet vulnerable to logic flaws or misuse. DeFi refers to a suite of financial applications built on top of blockchains, aiming to provide open, permissionless alternatives to traditional financial services [30]. From lending protocols and decentralized exchanges (DEXs) [27] to synthetic assets and derivatives [38], DeFi platforms have rapidly grown in complexity and adoption. With billions of dollars [3] locked in DeFi contracts, the ecosystem has become an attractive target for adversaries. Unlike traditional systems, where centralized oversight might limit damage or provide rollback mechanisms, DeFi attacks are typically fast, irreversible, and often replicated across chains–exposing a critical need for timely and generalizable vulnerability detection.

## 2.2 DeFi Attacks and Existing Detections

DeFi attacks fall broadly into two categories: price-driven [31, 46] and non-price [2] attacks. Price-driven attacks manipulate pricing mechanisms, such as oracles or liquidity pools, to buy low and sell high, extracting universal tokens (e.g., ETH [29], stable-coins) at favorable rates. These often involve flash loans or arbitrage and depend heavily on capital. Tools like POMABuster [46] and DeFiRanger [44] are tailored to detect such attacks by identifying abnormal value flows or predefined trading patterns, but their reliance on price-based signals makes them ill-suited for other exploit types.

Non-price attacks, on the other hand, exploit flaws in smart contract logic or protocol design, such as reentrancy [28], accounting errors [50], or privilege misconfigurations [45], without requiring capital. These exploits are diverse, subtle, and often obfuscated, making detection via hardcoded heuristics fragile. Tools like TxSpector [51] propose handcrafted rules for specific patterns (e.g., unchecked calls or suicidal contracts) but lack breadth. Forta [24], though proactive, relies on bytecode-level features that are sensitive to compiler noise, coding style, and obfuscation, and fail to generalize. While FlashGuard [2] recognizes the importance of non-price exploits, it focuses on post-incident mitigation rather than timely detection. To date, no existing tool offers a robust and scalable solution for semantically detecting diverse non-price attacks, highlighting the need for generalizable approaches like ours that reason over attacker intent at the trace level.

## 2.3 Motivation Example

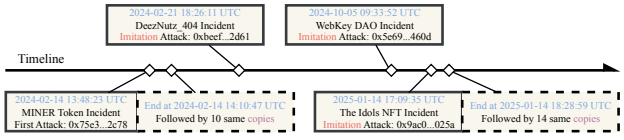



**Figure 1: MINER Token Incident and Imitative Attack Cascade. *Dashed boxes indicate repeated attacks from the same address ("Copies"), while "Imitation" refers to similar logic used by different addresses.***

Recent DeFi incidents reveal a recurring pattern: once an exploit is discovered, it often leads to a series of *imitative attacks* over time. While industry tools [26] sometimes detect the initial attack using ad hoc signals–such as exploiter addresses, Tornado [33] activity, or large transfers–these detections are often incidental. Without understanding the underlying exploit logic, they struggle to capture subsequent variants. To better illustrate the limitations of current detection methodologies, we present a representative example in Figure 1–a non-price-driven exploit that occurred on February 14, 2024. The attacker leveraged a subtle business logic flaw: the contract failed to prohibit self-transfers, and a temporary balance variable was mistakenly used for withdrawal validation. This allowed the attacker to withdraw more funds than permitted without relying on any price manipulation or initial capital [35]. Alarmingly, even after the incident was publicly disclosed, we continued to observe similar attacks exploiting nearly identical flaws well into 2025 [16]. This long tail of incidents demonstrates the failure of existing reactive systems to generalize detections across such *imitative attacks cascade*. From the Phalcon [18] and DefiHackLab [42] incident reports, we find that over 750 attacks in the past three years are repetitions of previously disclosed exploits, with more than 37% of the initial attacks having spawned 2-10 imitative copies. The cumulative loss attributed to these *imitative attacks* amounts to $2 billion.

# 3 Problem Definition and Overview

## 3.1 Problem Definition

The observations above (§2.3) motivate our research goal: to **generalize rapid, reactive detection for *imitative attack cascades*** once an initial attack instance is identified. To specify the problem, we introduce two concepts, *Unobserved Attack* (Definition 1) and *Reactive Detection Generalization* (Definition 2), that formalize our system's design goal.

Considering the case of the motivation example in Figure 1, where the initial exploit goes undetected by existing tools due to its novel logic. We define such an instance as an *Unobserved Attack*:

Definition 1 (Unobserved Attack). *Let $\mathcal{T}$ denote the universe of transactions, $\mathcal{A} \subseteq \mathcal{T}$ the set of attack transactions, $\mathcal{N} = \mathcal{T} \setminus \mathcal{A}$ the set of benign transactions, and $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ the set of deterministic detection patterns predefined where $p_i : \mathcal{T} \to \{True, False\}$. We define the set of* Observed Attacks *under pattern set $\mathcal{P}$ as:*

$$\mathcal{A}_{ob}(\mathcal{P}) = \left\{ t \in \mathcal{A} \;\middle|\; \begin{array}{l} \exists p_i \in \mathcal{P},\ p_i(t) = True \\ and\ \forall n \in \mathcal{N},\ p_i(n) = False \end{array} \right\}$$

*Then, the set of* Unobserved Attacks *is:*

$$\mathcal{A}_{unob}(\mathcal{P}) = \mathcal{A} \setminus \mathcal{A}_{ob}(\mathcal{P})$$

We say that a transaction $t$ is an *Unobserved Attack* (with respect to $\mathcal{P}$) if $t \in \mathcal{A}_{\text{unob}}(\mathcal{P})$. In other words, $t$ is an attack transaction that fails to be matched by any known detection rules implemented in current tools. To detect the *Unobserved Attacks* to prevent subsequent damage, our goal is to *Generalize Reactive Detection* for the *Unobserved Attack*:

Definition 2 (Reactive Detection Generalization). *Given an* Unobserved Attack *transaction $t \in \mathcal{A}_{unob}(\mathcal{P})$ with respect to a*

*pattern set $\mathcal{P}$, a* Reactive Detection Generalization *procedure constructs a new pattern $p^*$ such that:*

$$\forall n \in \mathcal{N}, p^*(t) = True \wedge p^*(n) = False$$

*Consequently, the transaction $t$ becomes detectable under the updated pattern set $\mathcal{P}' = \mathcal{P} \cup \{p^*\}$, i.e., $t \in \mathcal{A}_{ob}(\mathcal{P}')$.*

The definitions above describe idealized pattern sets $\mathcal{P}$ and $\mathcal{P}'$ under the assumption of perfect precision. In practical implementations, this requirement is relaxed to tolerate minor false positives, provided that the overall detection performance remains acceptable. Nevertheless, achieving *Reactive Detection Generalization* remains challenging for security experts due to (C1) the difficulty of abstracting diverse function semantics and (C2) matching logic in noisy, obfuscated traces. Manual analysis delays response and gives attackers further opportunities to act, highlighting the need for automated solutions to accelerate reaction time.

## 3.2 Overview of GenDetect

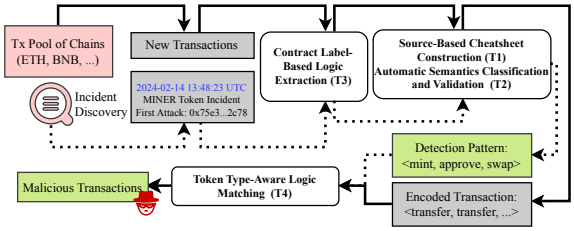



**Figure 2: The Overview of GenDetect**

**Overall Design.** With our goal formally defined, we now outline the overall design of *GenDetect*, which operationalizes the path from challenge to insight to implementation. As shown in Figure 2, the system takes two inputs: ① a newly discovered attack incident, which serves as a starting point for *reactive detection generalization*, and ② a pool of pending transactions to be analyzed. To address the challenges introduced earlier, GenDetect adopts two key insights: **Source-based Semantic Extraction** and **Contract Label-based Logic Matching**. These insights are instantiated through four core components: I) Technique 1 and 2 implement Insight I by transforming raw transaction traces into semantically meaningful representations via contract source code analysis; II) Technique 3 and 4 realize Insight II by refining the trace into an abstracted logic path and matching it against the pattern.

**Solving the Challenges.** The first challenge stems from the difficulty in accurately inferring the semantics of DeFi-specific function signatures (C1). Existing methods typically rely on word-level embeddings to classify function names. This works in some cases, e.g., `swapExactTokens`, `swap`, and `transferFrom` can be reasonably grouped with `swap` and `transfer` due to shared intent. However, in other cases, this approach clearly fails, e.g., `deposit` is different from `beforeDeposit`. Despite similar word-level tokens, these functions often have entirely different roles: `deposit` triggers a financial operation, whereas `beforeDeposit` merely performs access checks. Such semantic misclassification undermines the reliability of transaction-level analysis.

We tackle this issue by anchoring our solution in a simple yet powerful insight: only source code can faithfully reflect a function's actual behavior. Fortunately, most valuable DeFi projects are open source. We capitalize on this property with a two-stage design. First, we propose **T1: Source-Based Cheatsheet Construction (§4.1 Figure 3)**. We first use CodeBERT to roughly cluster function signatures in popular DeFi codebases then manually inspect them by real-world functionality. Each signature is annotated based on its corresponding financial intent (e.g., swap, transfer, borrow), forming a curated reference table. This cheatsheet enables accurate, human-aligned classification for all known signatures. However, this alone is insufficient for full coverage. As new contracts accompanied with new function signatures are constantly emerging. To support unseen signatures, we introduce **T2: Automatic Semantics Classification and Validation (§4.1 Figure 4)**, which automatically fetches open-source code corresponding to new functions, uses CodeBERT to derive the most similar function stored in codebase, and then leverages LLM-based reasoning to validate the derived label. This hybrid strategy ensures our system remains robust and up-to-date, extending semantic understanding to novel cases in the wild.

The second challenge (C2) stems from the inherent complexity of transaction logic in DeFi attacks. Unlike malware detection or contract vulnerability analysis, there exists no standard approach for extracting and comparing the semantic intent of transaction traces. Existing tools often fall back on naive pattern matching techniques such as Longest Common Subsequence (LCS). However, naive LCS suffers from two key limitations: (i) It treats all function calls equally, failing to isolate security-critical actions from irrelevant or benign operations. As a result, traces of long, legitimate transactions dilute the attack signal, leading to false positives, and (ii) LCS is highly sensitive to the execution order. Many malicious transactions may follow the same logical steps as known attacks but in a different order, which reduces similarity scores and increases false negatives.

To address this, we observe that the essence of an attack is often concentrated in a small subset of operations initiated by the attacker and targeting two typical categories of DeFi protocols. Therefore, the solution lies in selectively extracting meaningful logic and applying similarity analysis that is robust to noise. We realize this insight through two steps. **T3: Contract Label-based Logic Extraction (§4.2 Figure 5)**. Leveraging publicly available Contract Labeling datasets (Phalcon and 4Bytes) and transaction metadata, we identify key participants such as DeFi protocol addresses and attacker-controlled contracts. We then prune the trace by filtering out internal or routine contract invocations and collapsing nested layers into a flattened trace that preserves only the essential operations. This drastically reduces irrelevant noise and improves the signal-to-noise ratio for subsequent analysis. **T4: Token Type-aware Logic Matching (§4.3 Figure 6)**. The extracted logic is further partitioned based on the target token's economic role, distinguishing between core asset tokens (e.g., WETH, USDC) and protocol-specific tokens (e.g., DOGE, AAVE). These represent different attack intents: value extraction versus exploitation. For each token category, we compute the "Asymmetrical Normalized Set Difference (ANSD)" between traces, then aggregate weighted scores. This strategy boosts robustness in two ways: (1) benign

traces tend to focus on a single category, while attacks involve both, reducing false positives; (2) using asymmetrical set-based rather than sequence-based comparison mitigates false negatives caused by minor reordering and inserted noise.

## 4 The GenDetect

Having introduced our two core challenges and the key insights that guided our solution design, we now turn to the concrete technical realization of these four techniques.

### 4.1 Source-based Semantic Extraction

**Function Signature Cheatsheet Implementation.** To construct our Function Signature Cheatsheet, we begin by collecting historical attack data from DeFiHackLab [42] and Phalcon [18], extracting all function signatures involved in security incidents over the past five years. This process yields 1,272 unique function signatures linked to DeFi protocol projects. Next, using the "Decoded Projects" dataset provided by Dune [8], we successfully indexed the corresponding implementation code for each of these functions. This allows us to build a comprehensive source code database.

We then apply CodeBERT [22] to compute pairwise code-level semantic similarities across all collected function source code. For clustering, we adopt a K-means-based approach with an initial setting of $K = 80$, chosen heuristically based on functional diversity. The computation took approximately 3 hours. Following clustering, we manually validated and refined the results. The final cheatsheet was expanded to contain 122 distinct function categories. Among them: ① 60 categories correspond to high-frequency financial or verification-related functions, which appeared at least twice across different attacks. These include core operations such as `transfer`, `swap`, `mint`, as well as verification hooks like `beforeDeposit` and `afterBorrow`. ② The remaining 62 categories represent low-frequency or project-specific functions, each called fewer than twice across five years of attack data. These include niche operations such as `emergencyBurn`, `burnAlp`, or `airDropReward`. This cheatsheet enables us to instantly map a known signature to its semantic category whenever encountered in future traces, significantly improving trace abstraction efficiency. Figure 3 provides a brief illustration of the process.

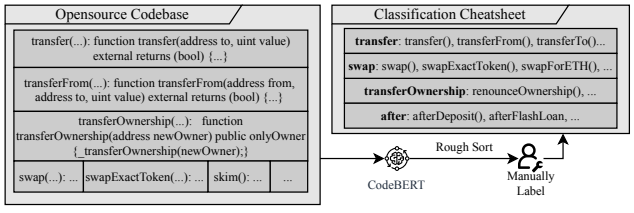



**Figure 3: Function Signature Cheatsheet Construction (T1).** ***The left is the CodeBase, a mapping from function signatures to their implementation code. The right is the final Function Cheatsheet, where each category includes multiple semantically similar function signatures. Importantly, clustering is based on source code, not just naming conventions.***

**Automatic Pipeline Implementation.** To handle unseen function signatures that are not covered by our cheatsheet, we introduce an Automatic Semantics Classification and Validation pipeline, as illustrated in Figure 4. For any signature already present in the cheatsheet, we can directly retrieve its semantic label. For those not covered, we follow a three-step fallback mechanism: ① Contract Decoding Check: We query Etherscan API [21] to check if the contract containing the unknown function has been decoded. If not, we discard it, as these are likely attacker-crafted and will be filtered out during logic extraction (see §4.2 Figure 5). ② Code Similarity Search: If decoded, we retrieve its implementation code and compare it against a reduced reference codebase $\text{CodeBase}^{(o)}$, which is a curated subset of representative implementations from each cheatsheet category. This optimization significantly reduces the computational cost of CodeBERT-based similarity matching. ③ Validation via GPT-4.1: Once the most similar category is identified, we perform a final heuristic validation using GPT-4.1. If GPT confirms the categorization, the function is assigned to that category; otherwise, we create a new class in the cheatsheet for future reference. This layered fallback ensures robust semantic coverage, even for novel or rarely used DeFi functions that are not part of the initial cheatsheet database.

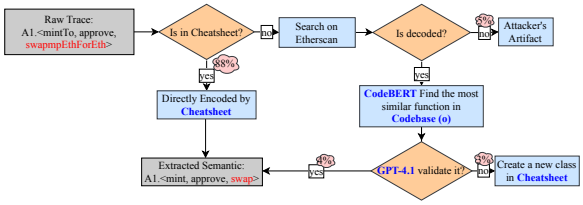



**Figure 4: Automatic Semantic Classification and Validation (T2).** ***The red cloud icons indicates the relative proportion of data processed along each branch of the pipeline.***

### 4.2 Contract Label-based Logic Extraction

**Address Labelling.** We illustrate the logic extraction process in Figure 5. The first step involves tagging each address invocation in the trace. Notably, most well-known or high-value project addresses are labeled in the 4Bytes [1] database. For less mainstream or newer project addresses, we rely on labels provided by the Phalcon API [18]. While we cannot fully eliminate the possibility that some recently deployed projects may not yet be included in these datasets, we note that our system's accuracy can be further improved as the labeling coverage of community-maintained datasets increases. Addresses that lack labels, or are explicitly marked as exploiter, or are directly called by the transaction sender, are grouped into the category `AttackerScript`. These will serve as the primary focus for the next phase of logic extraction.

**Filtration.** In the second step, we retain only direct invocations from the sender or identified `AttackerScript`, as only these invocations reflect the attacker's intentional actions. Downstream calls that originate from the protocol itself (e.g., recursive contract calls or triggered hooks) are excluded because they represent protocol logic rather than attacker intent. Finally, we perform layer restructuring on the remaining invocations. Specifically, for any invocation made by `sender.call` or `AttackerScript.call` we remove the wrapper call itself and lift its internal subcalls one level up in the

hierarchy. This adjustment reflects the fact that the outer attacker-generated calls are often meaningless or not decoded, whereas the meaningful behavior is embedded in the downstream interactions with protocol contracts. By eliminating these superficial wrappers while preserving the original call depth of the subcalls, we obtain a cleaner and more interpretable Extracted Logic that better captures the attacker's true intent.

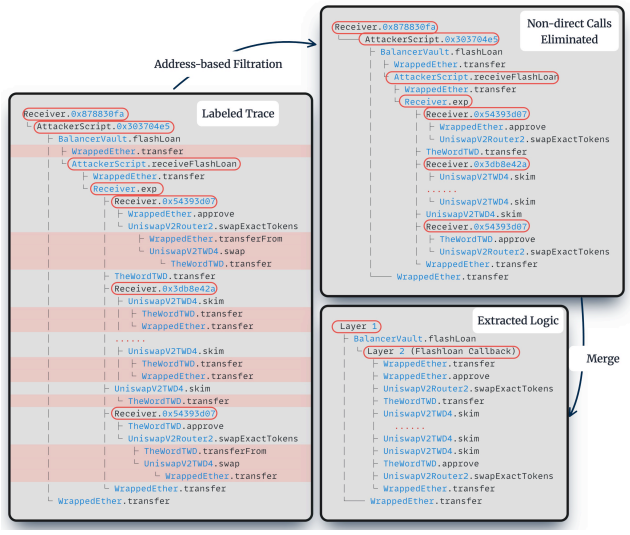



**Figure 5: Demonstration of Address-based Logic Extraction (T3).**

## 4.3 Token Type-Aware Logic Matching

**Type Labelling.** This module enhances the accuracy of logic similarity by introducing token type awareness into the matching process. Specifically, we classify each invocation in the Extracted Logic (output of §4.2) into two categories based on the token involved: ① Core Asset Operations: Interactions with widely-used tokens such as ETH, BNB, USDT, etc. ② Protocol-specific Operations: Interactions with tokens that are unique to a particular protocol or newly deployed projects, such as DOGE derivatives or bespoke tokens like RisyToken. As illustrated in the green box of Figure 6, different token types are marked with distinct colors, e.g., blue for core assets and black for protocol-specific assets. Next, as shown in the red box, given two transactions A and B, we compute two separate similarity scores: (i) $\text{Sim}_{\text{core}}(A, B)$: similarity between core asset operations in A and B and (ii) $\text{Sim}_{\text{proto}}(A, B)$: similarity between protocol-specific operations in A and B. The final similarity score is then computed as a weighted average of the two: $\text{Sim}_{\text{final}}(A, B) = \lambda \cdot \text{Sim}_{\text{core}} + (1 - \lambda) \cdot \text{Sim}_{\text{proto}}$

This token-type-aware decomposition improves the contrast between benign and malicious behavior. Benign transactions tend to contain only one category of operations, while imitative attacks often span both, making this split particularly effective at reducing false positives (FP). An illustrative example demonstrating the benefit of this separation will be presented in the case study (Figure 7).

**Asymmetrical Normalized Set Difference.** To compute the similarity between two extracted logic sequences, we adopt a set-based approach rather than a sequence-based one like LCS (Longest Common Subsequence). This choice is motivated by the observation that imitative attacks frequently involve logic mutation: while the core behaviors remain similar, their invocation order and auxiliary operations often differ. Sequence-based methods like LCS are sensitive to such ordering variations and thus perform poorly in this context. In contrast, set-based methods are more robust to such transformations.

We use a metric called Asymmetrical Normalized Set Difference (ANSD) for similarity calculation. Given two transactions $A$ and $B$, we define the similarity from $B$ to $A$ as: $\text{Sim}(A, B) = 1 - \frac{|A \setminus B|}{|A|}$. This expression measures how much of the reference logic $A$ (which is a confirmed attack) is preserved in $B$. Based on this metric, our detection rule is straightforward: if the final similarity score $\text{Sim}_{\text{final}}(A, B) \geq \tau$ ($\tau$ is a predefined threshold), we flag transaction $B$ as an imitative attack.

**Rationale for ANSD Design.** The rationale for the asymmetric design is twofold:

*(i) Why is the denominator $|A|$, rather than $|B|$, $|A \cup B|$?* Since $A$ is a verified attack pattern, it serves as the ground truth. Our goal is to assess whether $B$ contains all the necessary elements to reproduce the same exploit. $A$ denominator of $|A|$ ensures that the metric reflects the proportion of the attack pattern retained.

*(ii) Why use $A \setminus B$ as the numerator?* We assume that attack pattern $A$ is sufficient but may not minimal: any transaction that omits elements from $A$ is less likely to constitute the same threat. Hence, the more elements from $A$ that are missing in $B$, the lower the similarity. In contrast, using $B \setminus A$ would penalize irrelevant additions, which may be benign noise rather than part of the exploit, and is therefore less meaningful in this context.

By using the confirmed attack $A$ as the reference, the metric disregards any additional noise or evasion operations introduced in $B$, ensuring that only the omission of logic from $A$ leads to a lower similarity score. This design makes evasion behaviors, such as adding misleading calls, largely ineffective, reducing false negatives (FN). A example illustrating this robustness is provided in Figure 7.

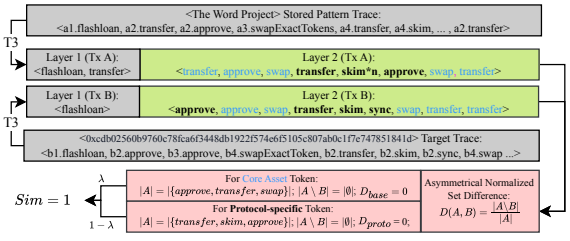



**Figure 6: Demonstration of Type-aware Logic Similarity Matching (T4).**

**Case Study: Filtering Benign Transactions and Tolerating Mutations.** We present three illustrative cases in Figure 7 to highlight the advantages of type-aware matching and our Asymmetrical Normalized Set Difference (ANSD) metric. The first two in green boxes are benign transactions: ① on core assets; ② on protocol-specific assets. In both cases, the non-type-aware method yields inflated similarity scores, while the type-aware variant correctly

down-weights them, based on our insight that transactions involving only one type of token operation are unlikely to reproduce the full logic of an attack. The third example is an evasion attempt: ③ it mutates call order (`approve`) and adds irrelevant noise (`buy`, `sell`). Traditional sequence-based methods like LCS are sensitive to such changes and produce low similarity, whereas our ANSD remains stable by focusing on functional coverage and ignoring order and noise. These cases validate that type-aware ANSD improves discriminative power, suppressing false positives and maintaining robustness under adversarial mutations.

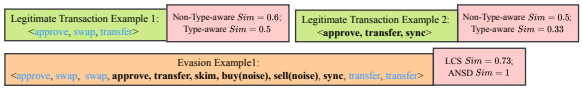



**Figure 7: Benign vs. Evasive Transactions: Similarity Analysis**

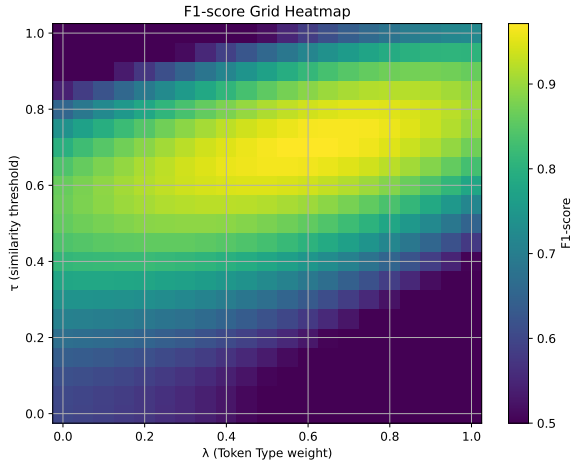



**Figure 8: Hyperparameter($[\lambda, \tau]$) Optimization through Nested 4-fold Cross-validation**

**Hyperparameter Optimization for Logic Matching.** The overall logic matching predicate is defined as:

$$p_A(B) = \begin{cases} \text{True}, & \text{if } \lambda \cdot \text{Sim}_{\text{core}} + (1-\lambda) \cdot \text{Sim}_{\text{proto}} \geq \tau \\ \text{False}, & \text{otherwise} \end{cases}$$

, where $\lambda \in [0, 1]$ adjusts the contribution between core asset type and protocol-specific type operation, and $\tau \in [0, 1]$ adjusts the final similarity threshold to determine whether a transaction is flagged as malicious. We tune both hyperparameters $(\lambda, \tau)$ using grid search on a held-out validation split within each fold during nested 4-fold cross-validation (Figure 8). Specifically, for each of the four folds, we split the training portion (801 samples from 534 attack and 534 benign examples) into a "9:1" internal split: approximately 720 transactions for training and 81 for validation. The hyperparameters are selected based on the F1-score performance on this internal validation set. This strategy ensures that the outer test folds remain completely untouched during tuning, preserving the integrity and fairness of cross-validation evaluation. The final parameter ranges ($\lambda \in [0.52, 0.66]$, $\tau \in [0.68, 0.72]$) correspond to the high-performing region highlighted in Figure 8.

## 5 Evaluation

We evaluate GenDetect by answering the following questions:
**RQ1:** How accurate is our detection framework, and how does each module contribute to its performance?
**RQ2:** Can our system discover new, previously undetected attacks in the wild?
**RQ3:** Is our system capable of operating under real-time constraints, and what is the computational cost of each core module?
**Baseline Tools.** We evaluate our system against four representative detection tools: TxSpector [51], POMABuster [46], DeFiRanger [44], and Forta [24]. TxSpector detects low-level logic bugs (e.g., reentrancy, suicidal contracts, unchecked calls) using handcrafted rules on decoded traces; we adopt the official implementation [34]. POMABuster targets financial exploits via pattern matching on funding flows, and we directly use its released code [39]. DeFiRanger extends this pattern-based strategy with additional rules, but only an unofficial implementation exists–released by the authors of POMABuster [39]–which covers only price-related patterns. To ensure a fair comparison, we supplement this baseline with our implementation of the missing non-price rules based on the original paper. Forta is a commercial monitoring framework based on contract bytecode; we use its open-source agent code [25] and retrain it on our benchmark. Unlike Forta, which requires training on bytecode, the other tools operate with fixed rules. Our system is trained on the same benchmark dataset as Forta for detection generalization and hyperparameter optimization.
**Experimental Environment.** All experiments were conducted on a Linux server running Ubuntu 24.04 LTS, equipped with an Intel Xeon Platinum 8268 CPU (96 logical cores at 2.90GHz) and 256 GB of RAM. Our implementation is developed in Python 3.11 and relies on the following major libraries: OpenAI API (openai 1.58.1), code-bert-score 0.4.1 and scikit-learn 1.5.1. All model inference and detection tasks were executed on the CPU only.

### 5.1 RQ1: Detection Accuracy

This section evaluates the detection accuracy of our system and baseline tools under both in-distribution and out-of-distribution (OOD) settings. We first report the cross-validation performance of all compared methods on the benchmark dataset, including the results of our ablation variants (Table 1). Then we evaluate it under skewed ratios and mixed category settings (Table 2). Next, we evaluate the zero-shot generalization ability of each system using the held-out dataset. To avoid potential data leakage and ensure a fair evaluation, we perform cross-domain evaluation by swapping the training and test sources, i.e., using DeFiHackLab for training and Phalcon for testing, and vice versa (Table 3). These evaluation setups allow us to assess GenDetect's detection rate across real-world DeFi attack scenarios.
**Setup.** As a preliminary evaluation, we use 4-fold cross-validation to evaluate all methods on a balanced benchmark dataset consisting of 534 malicious traces from DeFiHackLab [42], and 534 benign traces sampled from decentralized exchange (DEX) transactions recorded on the Dune platform [8]. To supplement its performance

**Table 1: 4-Fold Cross-Validation Results (Mean ± Std) Across Detection Methods and Ablations on DEX-Only Dataset. *w/o = without, SX = Semantics Extraction, LX = Logic Extraction, TTA = Token Type-aware, LCS/ANSD = replace Asymmetrical Normalized Set Difference with Longest Common Subsequence***

| Method | F1-score | FPR | FNR | Accuracy | Recall |
|---|---|---|---|---|---|
| **GenDetect** | **.98 ± .01** | **.01 ± .01** | **.03 ± .01** | **.98 ± .01** | **.97 ± .01** |
| w/o SX | .42 ± .15 | .44 ± .06 | .61 ± .15 | .48 ± .11 | .39 ± .15 |
| w/o LX | .72 ± .02 | .22 ± .08 | .30 ± .06 | .74 ± .02 | .70 ± .06 |
| w/o TTA | .90 ± .02 | .17 ± .04 | .03 ± .01 | .89 ± .02 | .97 ± .01 |
| LCS/ANSD | .91 ± .03 | .01 ± .01 | .13 ± .05 | .92 ± .03 | .87 ± .05 |
| Forta | .91 ± .02 | .16 ± .03 | .04 ± .02 | .90 ± .02 | .96 ± .02 |
| DefiRanger | .77 ± .06 | .23 ± .03 | .22 ± .09 | .77 ± .05 | .78 ± .09 |

**Table 2: Cross-Validation Results of GenDetect under Skewed Malicious:Benign (M:B) Ratios on DEX-Only Dataset and Mixed Dataset including Aggregator, Staking, DEX, and NFT Transactions.**

| M:B Ratio | F1-score(%) | FPR(%) | FNR(%) | Accuracy(%) | Recall(%) |
|---|---|---|---|---|---|
| 1:1 (DEX) | 98.0 ± 1.1 | 0.9 ± 0.9 | 3.1 ± 1.4 | 98.0 ± 1.1 | 96.9 ± 1.4 |
| 1:5 (DEX) | 97.6 ± 1.1 | 2.3 ± 1.2 | 2.3 ± 0.9 | 97.6 ± 1.1 | 97.6 ± 0.9 |
| 1:25 (DEX) | 95.7 ± 1.4 | 3.8 ± 1.3 | 4.6 ± 1.6 | 95.8 ± 1.4 | 95.4 ± 1.6 |
| 1:1 (Mix) | 98.6 ± 0.9 | 0.0 ± 0.0 | 2.6 ± 1.7 | 98.7 ± 0.8 | 97.3 ± 1.7 |
| 1:5 (Mix) | 98.3 ± 0.8 | 0.0 ± 0.0 | 2.9 ± 1.7 | 99.5 ± 0.2 | 97.1 ± 1.7 |
| 1:25 (Mix) | 97.9 ± 0.6 | 0.0 ± 0.0 | 3.7 ± 1.5 | 99.7 ± 0.0 | 96.2 ± 1.5 |

under real-world data distributions, we further conduct evaluation on a mixed dataset containing 13,300 transactions from various DeFi categories. It proportionally includes DEX [5], Aggregator [4], Staking [7], and NFT [6] contracts. Beyond the standard balanced (1:1) setting, we also evaluate the robustness of GenDetect under skewed malicious-to-benign (M:B) ratios of 1:5 and 1:25 on the mixed dataset, reflecting real-world scenario where benign transactions outnumber malicious ones. For tools other than Forta, we directly use decoded transaction traces as input, as they are capable of processing features derived from execution traces. However, Forta only operates on deployed smart contract. To accommodate this input format, we extract the corresponding contract code for each relevant transaction from data sources and construct a parallel input set for Forta's evaluation and training.

In the **cross-validation setting**, we compare GenDetect with Forta and DeFiRanger. We exclude POMABuster and TxSpector due to their narrow scope: the former focuses solely on price oracle manipulation, while the latter targets a small set of low-level bugs like reentrancy and unchecked calls. Their limited coverage restricts their value in broader benchmark comparisons. In contrast, Forta and DeFiRanger capture higher-level behavioral patterns and support a wider range of attack types, making them more suitable for diverse evaluation. In the **zero-shot setting** on the Phalcon and DeFiHackLab dataset [18, 42], we conduct cross-domain evaluation, i.e., using DeFiHackLab for training and Phalcon for testing, and vice versa. This setup further extinguish the potential of data leakage caused by usage of address label (§4.2) provided by Phalcon. We include all five tools: GenDetect, Forta, DeFiRanger, TxSpector, and POMABuster to assess performance on previously unseen attacks. Notably, while GenDetect and Forta require training on the benchmark dataset, the detection rules of POMABuster, DeFiRanger, and TxSpector are fixed, illustrating a key distinction between trained and pattern-based systems.

**Cross-Validation Results.** As shown in Table 1, GenDetect significantly outperforms both Forta and DeFiRanger across all major evaluation metrics on the cross-validation benchmark. Specifically, we achieve an F1-score of **0.98**, with a false positive rate (FPR) of only **0.01**. In contrast, Forta exhibits over **16**× higher FPR, while DeFiRanger fails to capture a substantial portion of semantic attacks, resulting in an F1-score of just **0.63**. In Table 2, with ratio changing from 1:1 to 1:25, the F1-score decreases by 0.8%, and FNR increases by 1%. GenDetect maintains consistent and reliable performance even when benign transactions substantially outnumber malicious ones.

*1) Understanding Ablation Study.* To further understand the contribution of each design component. The **ablation results** show that removing any single module leads to a considerable drop in performance. Removing the semantics extraction module (w/o SX) leads to the most severe degradation, with the F1-score dropping over 50% and the false negative rate (FNR) increasing to 0.61. This highlights the critical role of source-based semantic grouping in recognizing attack behaviors that would otherwise be misclassified or missed entirely. Excluding the logic extraction component (w/o LX) results in moderate performance degradation (F1 = 0.72), primarily due to increased noise from irrelevant invocations, which elevates both false positive and false negative rates. Disabling the token type-aware matching (w/o TTA) yields a relatively smaller drop (F1 = 0.90), with a noticeable increase in false positives, indicating its effectiveness in improving discrimination between benign and malicious traces. When replacing our set-based similarity metric with the sequence-sensitive LCS algorithm (LCS/ANSD), the FNR rises sharply to 0.13, confirming our hypothesis that LCS is vulnerable to minor mutations or reordering within transactions.

*2) Impact of Dataset Categories.* To examine the robustness of GenDetect across different categories of transactions, we evaluate its performance on a mixed-categories dataset (Table 2). The results show that FPR and FNR slightly decreased in the mixed setting compared to DEX-only setting. This improvement is due to the structural characteristics of the mixed dataset: unlike DEX protocols, which often involve long and complex multi-hop swap spanning multiple contracts, other application types such as NFT marketplaces typically feature shorter and more deterministic transaction patterns, e.g., single buy, sell, or approve calls. And staking systems mainly involve straightforward deposit or withdraw operations with limited contract depth. These short and semantically consistent transaction traces are easier to distinguish from malicious behaviors, thereby slightly lowering the overall error rates.

*3) Impact of Skewed Data Ratio.* Table 2 shows GenDetect's performance on datasets with progressively skewed malicious-to-benign ratios. We observe that as the proportion of benign transactions increases, FPR and FNR rises slightly. Since the training process of hyperparameters is based on the benign-dominant distribution, it slightly increases the decision threshold $\tau$ (§4.3) for detecting malicious traces, leading to a few attacks being missed. Meanwhile, with more benign samples in the test set, the absolute number of false positives also increases marginally. But the overall effect remains minimal, and GenDetect maintains stable performance even under highly imbalanced conditions.

*4) Understanding Baseline Tools' Limitations.* Forta reports high accuracy on its benchmark [24], but performs significantly worse in our setting. This discrepancy is primarily due to its detection paradigm: Forta relies on features extracted from attacker contracts themselves, assuming either source code availability or strong bytecode patterns. However, in real-world settings, most attacker contracts are unverified, obfuscated, or inaccessible, limiting the effectiveness of contract-level detection. In contrast, our method does not require visibility into the attacker's contract implementation. Instead, we analyze execution traces and focus on decoded, labeled protocol interactions to derive the attacker's intent. This trace-level semantic modeling allows us to generalize more effectively under realistic conditions with low attack contract transparency, explaining the superior performance of GenDetect on real-world attacks. DeFiRanger, while incorporating richer patterns than low-level tools like TxSpector, still relies heavily on exact rule matching and fragile execution signatures. This leads to high false negative rates when faced with semantically similar but structurally variant attacks. Its limited generalization stems from overfitting to specific symbolic patterns, whereas our semantic encoding enables more robust behavioral abstraction, yielding superior coverage across diverse real-world exploits.

**Table 3: Summarized Cross-domain Evaluation: Zero-shot Detection Coverage on Phalcon or DeFiHackLab Attack Dataset. ✓ = *detected,* △ = *running error, blank = not detected.* (For full results, please see Github [11, 12])**

| Projects | GenDetect (Ablation) | | | | Forta | POMA | DeFiR | TxSpec |
|---|---|---|---|---|---|---|---|---|
| From Phalcon [18] | Full | w/o SX | w/o LX | w/o TTA | | | | |
| MIM_Spell | ✓ | | ✓ | | | △ | | |
| Zoth | | | | | | △ | △ | |
| 1inch Fusion V1 | ✓ | | ✓ | | ✓ | ✓ | ✓ | |
| Infini | | | | | | △ | △ | |
| HegicOptions | ✓ | ✓ | ✓ | | ✓ | ✓ | | |
| Bybit | | | | | | △ | △ | |
| More results: Repo [11] | ... | ... | ... | ... | ... | ... | ... | ... |
| **Coverage** | **.7656** | .2813 | .5625 | .6094 | .6563 | .2813 | .4375 | .3125 |
| From DeFiHackLab [42] | | | | | | | | |
| 0vix | ✓ | | ✓ | ✓ | ✓ | ✓ | | |
| 0x0DEX | ✓ | ✓ | ✓ | ✓ | ✓ | △ | △ | |
| AES_1 | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | |
| AES_2 | ✓ | ✓ | ✓ | | ✓ | ✓ | | |
| AIS | | | | | | | | |
| Allbridge | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | |
| More results:Repo [12] | ... | ... | ... | ... | ... | ... | ... | ... |
| **Coverage** | **.7284** | .3824 | .5009 | .5697 | .6481 | .2485 | .3613 | .1950 |

**Zero-Shot Results.** We evaluate the detection rate of all tools on held-out, real-world datasets consisting of 726 attack incidents reported by Phalcon [18] and DeFiHackLab [42]. Table 3 shows the summarized results (Github Repo [11, 12] for full results): GenDetect achieves a coverage rate of **77/73%**, significantly outperforming Forta (66/65%), DeFiRanger (44/36%), TxSpector (31/20%), and POMABuster (28/25%). Among our ablation variants, the model without the semantic extraction module covers only **28%-38%** of incidents, indicating a nearly **50%** drop in detection capacity. Removing either the logic extraction or token type-aware module reduces coverage to below 60%, reflecting 16-22% performance degradation.

*1) Understanding Detection Rate Drop.* As expected, detection performance in the zero-shot setting is lower than in the cross-validation experiments. This is largely due to the complete disjointness between the test set and the training set, which is a common source of generalization loss in real-world detection tasks. Nevertheless, our performance drop is moderate: compared to the detection rate (recall) reduction of 31-32% observed in Forta, our model exhibits only a 21-25% decline, suggesting better stability in the face of data in the wild. Upon closer inspection, we find that most of the undetected attacks fall into edge-case categories that lie outside the observable transaction semantics. In particular, several missed incidents involve **private key leakage**, where the exploit resembles a side-channel attack with no observable execution anomalies on-chain. Another group of hard-to-detect samples involves **unverified call input attacks**, where a single external call results in the entire asset drain, yet leaves behind little to no semantic trail. These cases are inherently difficult to capture with any trace-based or behavior-level approach, and we acknowledge them as current limitations of our design.

*2) Understanding Baseline Tools' Limitations.* Our model substantially outperforms all baseline tools under the zero-shot setting. Forta's performance is limited by its reliance on code-level features, which are sensitive to developer coding style and compiler optimizations and fail to robustly reflect attack intent. POMABuster, DeFiRanger, and TxSpector suffer from similar weaknesses: all are pattern-based systems with hardcoded heuristics, offering narrow coverage and low adaptability to novel attack variants. Furthermore, many transactions in the Phalcon dataset lack the specific signals required by these systems, e.g., POMABuster requires token price information that may be unavailable for unlisted assets, such as NFT-related transactions or those involving project-specific "internal token ledger" (i.e., with no publicly accessible price data) [16], leading to incomplete execution or detection failure.

*3) Potential Data Leakage and Resolution.* Since our method leverages manually annotated contract address labels from the Phalcon dataset, such label alignment may introduce a potential implicit advantage to the detection model, as both the labels and the transactions originate from the same source organization. To rule out potential data leakage from shared label sources, we perform a cross-domain evaluation using the independent DeFiHackLab dataset. Although the coverage slightly decreases from 76% to 73% (Table 3), the overall performance remains consistent, indicating that any implicit linkage between datasets does not affect GenDetect's usability or the validity of our conclusions.

## 5.2 RQ2: New Attack Discovery by GenDetect

In this section, we investigate the capability of our system to identify previously undisclosed attacks from real-world transaction data. We summarize the newly discovered attack instances in a comprehensive table on Github [9], with full details including transaction hashes, similarity scores, and matched references. This enables readers to independently inspect and validate the findings, also promotes transparency. We then analyze these results to understand their structural characteristics and to illustrate the effectiveness of our semantic similarity framework in capturing meaningful but unreported exploit behaviors.

**Setup.** To evaluate our system in a fully realistic setting, we conduct similarity-based detection on real Ethereum transaction data. In theory, the Ethereum mainnet has recorded over 1.5 billion transactions from January 2021 to December 2024. However, to make

the evaluation computationally feasible, we restrict our analysis to a targeted subset of transactions involving decentralized exchanges (DEXs), which are more likely to contain financially meaningful operations. This filtered dataset consists of approximately 3 million transactions, collected directly from the Dune Analytics platform [8]. The full detection pipeline was executed over this dataset in an end-to-end fashion, with a total processing time of approximately **77 hours**.

**Results.** Our full detection results include 682 public disclosed incidents, 56 original disclosure, 1862 MEV transactions (excluded from FP by definition) and 79 false positives, yielding an overall 3% FPR, consistent with cross-validation results. These non-original cases are listed in Github Repo [10]. To isolate our original disclosures, we applied a rigorous filtering pipeline. First, we excluded arbitrage and MEV-related transactions by analyzing sender history and existing MEV labels from Ethereum datasets[21]. Next, we filtered out known attack incidents using exploiter tags from both Ethereum and Phalcon [18]. The remaining candidates underwent manual validation based on strong indicators of exploitation: the use of Tornado Cash or cross-chain bridges for rapid fund exfiltration, temporary contract/account creation, and manual confirmation of vulnerabilities in the victim contracts. This conservative process yielded a final set of **56** previously undocumented attacks [9], with over **$1.5 million USD** in total financial damage. Unlike tools such as POMABuster, which report any suspicious transactions, our result prioritizes high-confidence real-world attacks with concrete behavioral evidence.

*Case Study and Reference.* Each representative attack case corresponds to a known historical exploit in DeFiHackLab [42]. For brevity, all full-length transaction hashes are listed in the Github Repo [9], with inline citations provided for verifiability. Among these, one notable category involves 10 previously unreported attacks that exhibit high structural similarity to the *pSeudoEth* incident. These attacks exploit a mispriced internal accounting mechanism by invoking `skim()` to redirect funds to the vault itself, resulting in unintended profit. Some individual gains were relatively modest, around $100, while others exceeded $3k, reflecting a wide spectrum of exploit severity. Despite this variation, the attacks exhibit consistent behavioral patterns, highlighting the systematic nature of the exploitation. These lower-value cases may have slipped under the radar of existing tools, emphasizing the strength of our approach: by analyzing core transactional logic rather than surface-level attributes, our system remains robust against obfuscation, randomness, and value-based evasion.

## 5.3 RQ3: Detection Efficiency

In this section, we evaluate the runtime efficiency of our system, with a focus on the two components most critical to real-time performance: the Semantic Cheatsheet Module and the Logic Extraction-Matching Module. We measure the per-transaction processing time of each module under varying trace lengths, demonstrating the practicality of our system in latency-sensitive settings. The other components in automatic semantic classification and validation pipeline are excluded from detailed timing analysis, because it is executed entirely offline and does not affect online performance. For completeness, we briefly summarize its expected cost profiles.

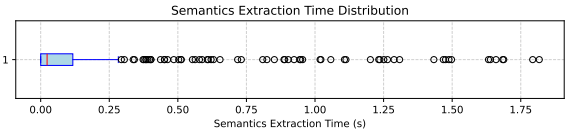



**Figure 9: Runtime Distribution of Semantic Extraction.** ***The box plot illustrates the latency distribution across all 534 attack traces, including median, quartiles, and outliers, reflecting per-transaction variance.***

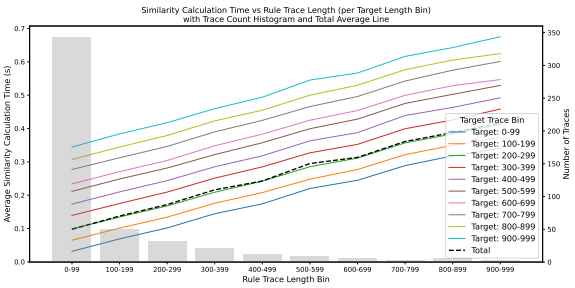



**Figure 10: Runtime of Logic Extraction and Matching under Varying Rule and Target Trace Lengths.** ***Bar Chart: the distribution of rule trace lengths in the dataset; Colored Lines: the average Logic Extraction and Matching time under a specific target trace length; Dashed Line: the global average latency across all target lengths for each rule length.***

**Setup.** To evaluate the runtime performance of our system, we measure the processing latency on a dataset of 534 attack transactions collected from DeFiHackLab [42]. These transactions represent real-world DeFi exploits and typically exhibit significantly more complex trace structures than regular user transactions. As such, the response times reported in this experiment reflect our system's behavior under **worst-case conditions**. The primary goal of this evaluation is to assess the feasibility of real-time detection in high-complexity scenarios.

**Results.** Our experiments reveal clear patterns in how runtime scales with trace complexity. For the Semantic Cheatsheet Module (Figure 9), the average per-transaction latency remains under **0.02s**, with only a few outliers exceeding **1.75s**. These rare cases typically involve highly irregular trace structures. For the Logic Module (Figure 10), both the length of the rule trace and the target trace jointly influence the runtime. In the worst-case configuration–where both the rule and the target reach the maximum trace length of 800–the observed latency remains bounded around **0.7s** and the global average runtime remains close to **0.1s**. These results align well with the theoretical complexity of the logic matching process, which operates at $O(M + N)$, where $M$ and $N$ denote the lengths of the rule and target traces, respectively.

*Real-Time Scenario Feasibility.* The true performance lower bound only manifests under extremely long traces–cases that are exceedingly rare in practice (<0.1%) and insufficient for statistical characterization. To assess real-time viability, we therefore conduct

an analysis using both average throughput and worst-case performance. ① For throughput, our system is capable of processing **over 2,000 transactions per second (TPS)** on 96 cores, based on Ethereum's [21] transaction length distribution. This throughput comfortably satisfies the real-time detection demands of most blockchain platforms, including Solana [40] under average load (1,133 TPS). While Solana's theoretical throughput limit is higher (65,000 TPS), our system is parallelizable by design and can be scaled horizontally to meet such high demands. ② Under worst-case performance, for instance, when Ethereum [21] (**12s**) and BNB [20] (**3s**) encounter one long-trace transaction, whose detection can be fully completed within **0.7s**. Since only the longest transactions contributes to the **critical path**, the remaining transactions can be processed in parallel on other cores, allowing total evaluation to finish well within the block interval. Hence, our system satisfies real-time requirements on these platforms. However, on Solana [40] (**0.4s**), GenDetect's worst-case cost slightly exceeds the block interval. We further discuss this limitation and potential solutions in §6.

## 6 Limitation and Discussion

Our approach presents two primary limitations: real-time performance on high-speed chains and evasion robustness.

**Performance.** While the method demonstrates low-latency detection per trace, meeting extreme throughput demands (e.g., Solana's 65K TPS) would require impractical horizontal scaling, approximately 30× more cores. A more feasible path is to pre-filter trivial transactions (e.g., those with $\leq 2$ calls), which make up 99.7% of total traffic. With only 86 long transactions per 0.4s block interval, and 0.1s average analysis time, we estimate that 22 cores suffice to meet practical real-time needs. Even for worst-case traces (0.7s), our Asymmetrical Normalized Set Difference (ANSD) algorithm supports parallel computation, where doubling the cores reduces worst-case latency to 0.35s, safely within a block interval.

**Potential Evasions.** Although GenDetect exhibits robustness against post-hoc mutation and noise, several potential evasion vectors remain. These can be categorized into three types below, each reflecting a different layer of the detection pipeline.

*First-instance Evasion:* Our contract-labelling and ANSD design mitigates most post-hoc evasions by filtering non-critical semantic mutations. However, if attackers anticipate our theory and inject obfuscation at the first occurrence of an attack pattern, the initial representation itself becomes noisy, degrading downstream similarity learning. Formalizing and defending against such first-instance evasions remain open challenges for current pattern-based systems.

*Label-compromise Evasion:* Because GenDetect relies on a manually curated address-label dataset, an attacker could theoretically evade detection by compromising the labelling process and having malicious addresses marked as benign. This would bypass address-based filtration. In practice, such side-channel manipulation is difficult since the label set is manually verified and periodically maintained, but it highlights the importance of data-source integrity.

*Private-relay Evasion:* Private Relay Network is originally introduced by Flashbots [23] to mitigate MEV by bypassing the public mempool. Some permissioned relays expose authenticated APIs that authorized institutions can monitor, allowing GenDetect to extend to these environments by incorporating relay or bundle data alongside public-mempool captures. However, fully closed private relays (often referred to as "dark-pools") pose a greater challenge. Transactions submitted exclusively through such relays are invisible to public-mempool monitors before inclusion, placing them outside the observable scope of GenDetect and other mempool-based detectors. Detecting attacks launched purely via closed private relays thus remains an open problem and represents an inherent limitation of observation-based detection. Future directions include deeper collaboration with relay operators, on-chain trace correlation, and new visibility instrumentation.

## 7 Related Work

A variety of tools have been developed to detect malicious transactions in DeFi, primarily following pattern-based or contract-level detection paradigms. Pattern-based systems such as DeFiRanger [44] and POMABuster [39] identify price-driven attacks by matching trade and oracle manipulation patterns, while TxSpector [34] targets logic bugs like reentrancy and unchecked external calls. Other domain-specific tools include LeiShen [48] for flashloan-related attacks and TLMG4Eth [41] for address-level fraud detection. Contract-level tools like Forta [24] analyze deployed bytecode to flag malicious behavior before execution, but face challenges due to compilation variance and limited semantic interpretability. Some works address post-exploit defense: APE [36] and Sting [52] simulate or mimic known attacks, while FlashGuard [2] mitigates non-price vulnerabilities after the fact. In contrast, our work focuses on generalizing detection patterns from traces to enable real-time, semantic-level detection of novel and polymorphic DeFi attacks.

## 8 Conclusion

This paper presents GenDetect, a detection generation framework designed to address the persistent challenge of recurring DeFi attacks through semantic generalization and structural similarity analysis. By abstracting execution traces into financial semantics and comparing them compositionally, our system captures the underlying intent of transactions rather than relying on brittle surface-level features. Its fully parallelizable per-transaction design–along with a set-based similarity metric (ANSD) that is inherently parallelizable–enables high-throughput performance, making it suitable for real-time deployment even under modern blockchain traffic. Our evaluation demonstrates its effectiveness across both benchmark and zero-shot settings, while uncovering previously unreported attacks in the wild. As DeFi continues to evolve, we believe GenDetect offers a robust foundation for scalable, interpretable, and timely threat detection, and we envision future work exploring enhanced semantic representations and accelerated similarity computation to further strengthen real-time defenses.

## 9 Acknowledgement

Kangjie Lu, Bowen Cai and Weiheng Bai were supported in part by NSF awards CNS-2045478, CNS-2106771, and CNS-2247434. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

## References


[1] 4bytes. 2025. ethereum-lists/contracts: List of contracts from known projects (work in progress). https://github.com/ethereum-lists/contracts

[2] Abdulrahman Alhaidari, Balaji Palanisamy, and Prashant Krishnamurthy. 2025. Protecting DeFi Platforms against Non-Price Flash Loan Attacks. *arXiv preprint arXiv:2503.01944* (2025).

[3] AliceHsu. [n. d.]. https://tech-blog.cymetrics.io/en/posts/alice/2024_defi_hack/

[4] Dune Analytics. 2025. Dune Dex Aggregator Trades Data. https://www.dune.com/data/dex_aggregator.trades

[5] Dune Analytics. 2025. Dune Dex Trades Data. https://www.dune.com/data/dex.trades

[6] Dune Analytics. 2025. Dune NFT Trades Data. https://www.dune.com/data/nft.trades

[7] Dune Analytics. 2025. Dune Staking Ethereum Flows Data. https://www.dune.com/data/staking_ethereum.flows

[8] Dune Analytics. n.d.. Dune: On-chain Crypto Data and Analytics. https://dune.com. Accessed April 19, 2025.

[9] Anonym. 2025. Anonymous Repository for Complementary New Discovered Attacks Data. https://github.com/NobodyIsAnonymous/ICSE2026Anonym/blob/main/discovered_attacks.pdf

[10] Anonym. 2025. Anonymous Repository for Complementary Original Results Data. https://github.com/NobodyIsAnonymous/ICSE2026Anonym/blob/main/original_results.txt

[11] Anonym. 2025. Anonymous Repository for Complementary Zero-shot Evaluation Data. https://github.com/NobodyIsAnonymous/ICSE2026Anonym/blob/main/zero-shot_evaluation.pdf

[12] Anonym. 2025. Anonymous Repository for Cross-domain Zero-shot Evaluation Data. https://github.com/NobodyIsAnonymous/ICSE2026Anonym/blob/main/DeFiHackLab_zero_test.csv

[13] Matteo Aquilina, Jon Frost, and Andreas Schrimpf. 2024. Decentralized finance (DeFi): a functional approach. *Journal of Financial Regulation* 10, 1 (2024), 1–27.

[14] BlockSec. [n. d.]. 0x5e694707337cca979d | Phalcon Explorer. https://app.blocksec.com

[15] BlockSec. [n. d.]. 0x75e3aeb00df69882a1 | Phalcon Explorer. https://app.blocksec.com

[16] BlockSec. [n. d.]. 0xa8289dbe3e49e9c5de | Phalcon Explorer. https://app.blocksec.com

[17] BlockSec. [n. d.]. 0xbeefd8faba2aa82704 | Phalcon Explorer. https://app.blocksec.com

[18] BlockSec. [n. d.]. Security Incidents | Phalcon Explorer. https://app.blocksec.com

[19] BlockSec. 2025. MetaSuites (formerly MetaDock): The Builder's Swiss Army Knife by BlockSec. https://blocksec.com/metasuites

[20] BscScan.com. 2025. BNB Smart Chain (BNB) Blockchain Explorer. https://bscscan.com/

[21] etherscan.io. [n. d.]. Ethereum (ETH) Blockchain Explorer. https://etherscan.io/

[22] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaodong Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. https://github.com/microsoft/CodeBERT. arXiv:2002.08155.

[23] Flashbots. 2025. Flashbots MEV-Boost. https://boost.flashbots.net

[24] Forta. [n. d.]. Forta. https://www.forta.org/

[25] Forta. 2025. forta-network/starter-kits. https://github.com/forta-network/starter-kits

[26] Krzysztof Gogol, Christian Killer, Malte Schlosser, Thomas Bocek, Burkhard Stiller, and Claudio Tessone. 2024. SoK: Decentralized Finance (DeFi)–Fundamentals, Taxonomy and Risks. *arXiv preprint arXiv:2404.11281* (2024).

[27] Sascha Hägele. 2024. Centralized exchanges vs. decentralized exchanges in cryptocurrency markets: A systematic literature review. *Electronic Markets* 34, 1 (2024), 33.

[28] Mingtao Ji, GuangJun Liang, Meng Li, Haoyan Zhang, and Jiacheng He. 2021. Security Analysis of Blockchain Smart Contract: Taking Reentrancy Vulnerability as an Example. In *Advances in Artificial Intelligence and Security: 7th International Conference, ICAIS 2021, Dublin, Ireland, July 19-23, 2021, Proceedings, Part III 7*. Springer, 492–501.

[29] Paraskevi Katsiampa. 2019. Volatility co-movement between Bitcoin and Ether. *Finance Research Letters* 30 (2019), 221–227.

[30] Iqbal Mabruri, Carles Cerqueda, Christopher Jack, Alexis Lui, Wenbin Wu, Vincezo DiPerna, Keith Bear, Bryan Zhang, et al. 2024. Dynamic Taxonomy a Bridge from DeFi to TradFi. *Available at SSRN* (2024).

[31] Yifan Mo, Jiachi Chen, Yanlin Wang, and Zibin Zheng. 2023. Toward automated detecting unanticipated price feed in smart contract. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 1257–1268.

[32] Roberto Moncada, Enrico Ferro, Alfredo Favenza, and Pierluigi Freni. 2021. Next generation blockchain-based financial services. In *Euro-Par 2020: Parallel processing workshops: Euro-Par 2020 international workshops, Warsaw, Poland, August 24–25, 2020, Revised selected papers 26*. Springer, 30–41.

[33] Matthias Nadler and Fabian Schär. 2023. Tornado cash and blockchain privacy: a primer for economists and policymakers. *Federal Reserve Bank of St. Louis Review* (2023).

[34] OSUSecLab. [n. d.]. OSUSecLab/TxSpector. https://github.com/OSUSecLab/TxSpector/tree/master

[35] Phalcon. [n. d.]. 0x75e3aeb00df69882a1 | Phalcon Explorer. https://app.blocksec.com

[36] Kaihua Qin, Stefanos Chaliasos, Liyi Zhou, Benjamin Livshits, Dawn Song, and Arthur Gervais. 2023. The blockchain imitation game. In *32nd USENIX Security Symposium (USENIX Security 23)*. 3961–3978.

[37] Muhammad Saad, Jeffrey Spaulding, Laurent Njilla, Charles Kamhoua, Sachin Shetty, DaeHun Nyang, and David Mohaisen. 2020. Exploring the attack surface of blockchain: A comprehensive survey. *IEEE Communications Surveys & Tutorials* 22, 3 (2020), 1977–2008.

[38] Fabian Schär. 2021. Decentralized finance: on blockchain and smart contract-based financial markets. *Review of the Federal Reserve Bank of St Louis* 103, 2 (2021), 153–174.

[39] Siriussee. [n. d.]. DependableSystemsLab/POMABuster: POMABuster is an automated engine to detect Price Oracle Manipualtion Attack (POMA) to blockchain oracles. https://github.com/DependableSystemsLab/POMABuster/tree/main

[40] solana. [n. d.]. Explorer | Solana. https://explorer.solana.com/

[41] Jianguo Sun, Yifan Jia, Yanbin Wang, Ye Tian, and Sheng Zhang. 2025. Ethereum fraud detection via joint transaction language model and graph representation learning. *Information Fusion* 120 (2025), 103074.

[42] SunWeb3Sec. 2025. SunWeb3Sec/DeFiHackLabs. https://github.com/SunWeb3Sec/DeFiHackLabs original-date: 2022-06-10T09:57:11Z.

[43] Shih-Hung Wang, Chia-Chien Wu, Yu-Chuan Liang, Li-Hsun Hsieh, and Hsu-Chun Hsiao. 2021. ProMutator: Detecting vulnerable price oracles in DeFi by mutated transactions. In *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 380–385.

[44] Siwei Wu, Zhou Yu, Dabao Wang, Yajin Zhou, Lei Wu, Haoyu Wang, and Xingliang Yuan. 2023. Defiranger: detecting DeFI price manipulation attacks. *IEEE Transactions on Dependable and Secure Computing* 21, 4 (2023), 4147–4161.

[45] Xiangyu Wu, Xuehui Du, Qiantao Yang, Aodi Liu, Na Wang, and Wenjuan Wang. 2023. TaintGuard: Preventing implicit privilege leakage in smart contract based on taint tracking at abstract syntax tree level. *Journal of Systems Architecture* 141 (2023), 102925.

[46] Rui Xi, Zehua Wang, and Karthik Pattabiraman. 2024. POMABuster: Detecting Price Oracle Manipulation Attacks in Decentralized Finance. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, IEEE, San Francisco, CA, USA, 3923–3942.

[47] Qing Xia, Zhirong Huang, Wensheng Dou, Yafeng Zhang, Fengjun Zhang, Geng Liang, and Chun Zuo. 2023. Detecting Flash Loan Based Attacks in Ethereum. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 154–165.

[48] Qing Xia, Zhirong Huang, Wensheng Dou, Yafeng Zhang, Fengjun Zhang, Geng Liang, and Chun Zuo. 2023. Detecting Flash Loan Based Attacks in Ethereum. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, IEEE, Online, 154–165.

[49] Teng Andrea Xu and Jiahua Xu. 2022. A short survey on business models of decentralized finance (DeFi) protocols. In *International Conference on Financial Cryptography and Data Security*. Springer, 197–206.

[50] Brian Zhang. 2024. Towards finding accounting errors in smart contracts. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[51] Mengya Zhang, Xiaokuan Zhang, Yinqian Zhang, and Zhiqiang Lin. 2020. {TXSPECTOR}: Uncovering attacks in ethereum from transactions. In *29th USENIX Security Symposium (USENIX Security 20)*. 2775–2792.

[52] Zhuo Zhang, Zhiqiang Lin, Marcelo Morales, Xiangyu Zhang, and Kaiyuan Zhang. 2023. Your exploit is mine: Instantly synthesizing counterattack smart contract. In *32nd USENIX Security Symposium (USENIX Security 23)*. 1757–1774.