Demystifying DeFi MEV Activities in Flashbots Bundle

Zihao Li
The Hong Kong Polytechnic
University, China

Jianfeng Li
Xi'an Jiaotong University, China

Zheyuan He
University of Electronic Science and
Technology of China, China

Xiapu Luo*
The Hong Kong Polytechnic
University, China

Ting Wang
Pennsylvania State University, USA

Xiaoze Ni
University of Electronic Science and
Technology of China, China

Wenwu Yang
University of Electronic Science and
Technology of China, China

Xi Chen
University of Electronic Science and
Technology of China, China

Ting Chen*
University of Electronic Science and
Technology of China, China

ABSTRACT

Decentralized Finance, mushrooming in permissionless blockchains, has attracted a recent surge in popularity. Due to the transparency of permissionless blockchains, opportunistic traders can compete to earn revenue by extracting Miner Extractable Value (MEV), which undermines both the consensus security and efficiency of blockchain systems. The Flashbots bundle mechanism further aggravates the MEV competition because it empowers opportunistic traders with the capability of designing more sophisticated MEV extraction. In this paper, we conduct the first systematic study on DeFi MEV activities in Flashbots bundle by developing ActLifter, a novel automated tool for accurately identifying DeFi actions in transactions of each bundle, and ActCluster, a new approach that leverages iterative clustering to facilitate us to discover known/unknown DeFi MEV activities. Extensive experimental results show that ActLifter can achieve nearly 100% precision and recall in DeFi action identification, significantly outperforming state-of-the-art techniques. Moreover, with the help of ActCluster, we obtain many new observations and discover 17 new kinds of DeFi MEV activities, which occur in 53.12% of bundles but have not been reported in existing studies.

CCS CONCEPTS

• **Security and privacy** → **Distributed systems security**.

KEYWORDS

DeFi, Smart contract, Miner Extractable Value, Flashbots Bundle

1 INTRODUCTION

Decentralized Finance (DeFi) has attracted a recent surge in popularity with more than 40B USD total locked value [10]. Since transactions broadcasted in the underlying P2P network of blockchain are globally visible, opportunistic traders can strategically adjust the gas price to prioritize their transactions and earn extra revenue from DeFi, which is known as the generic term Miner Extractable Value (MEV) [40, 44, 47, 72, 82, 96, 98].

MEV competition undermines both the security and efficiency of blockchain systems. First, it incentivizes financially rational validators (miners in the context of PoW) to fork the chain, thereby deteriorating the blockchain's consensus security [40, 72]. Second,

*Corresponding authors

it aggravates network congestion (i.e., P2P network load) and chain congestion (i.e., block space usage) because opportunistic traders who compete for MEV opportunities prioritize their transactions at the cost of considerable time delay for other transactions [40].

The Flashbots organization proposed the bundle mechanism which enables opportunistic traders to design more sophisticated MEV extraction for profits, because it allows traders to submit a sequence of self-constructed and/or selected transactions as a bundle, which can even include unconfirmed transactions broadcasted on the P2P network. It was reported that compared to the vanilla Sandwich attacks, the bundle-based variants were more profitable [20].

However, little is known about DeFi MEV activities conducted through the bundle mechanism. To demystify the status quo of DeFi MEV activities in bundles, we aim at answering the following questions, namely how prevalent are known DeFi MEV activities in bundles? Are there new DeFi MEV activities that are unreported before in bundles? If that is the case, how did they behave and how prevalent are they? What are the differences between DeFi MEV activities in bundles and other DeFi MEV activities? The answers to these questions can help researchers have an in-depth understanding of DeFi MEV activities, e.g., the features of various MEV activities and the robustness of today's MEV mitigation techniques.

In this paper, we conduct the first systematic study on DeFi MEV activities performed through Flashbots bundle. A DeFi MEV activity usually consists of several DeFi actions, each of which refers to an interaction between a trader and an individual function provided by the contracts of DeFi applications. For example, a contract of AMM (Automated Market Maker) should support the swap DeFi action for exchanging different assets [91]. A cyclic arbitrage [82] MEV activity involves multiple swap actions in different contracts of AMMs with different prices for profits.

To characterize DeFi MEV activities, we need to first recognize them according to their DeFi actions. Although existing studies [2, 18, 70–72, 82, 84, 85, 89] examined DeFi MEV activities and their DeFi actions, they cannot conduct a systematic study on DeFi MEV activities in Flashbots bundle because they suffer from two limitations. First, the majority of them [18, 70–72, 82, 84, 85] focus on a few DeFi applications and could not be easily extended to cover other DeFi applications because they rely on considerable manual efforts to derive the rules for recognizing DeFi actions according to the specific events emitted by the contracts of DeFi applications and their arguments (cf. Table 1). Thus, they will miss many DeFi
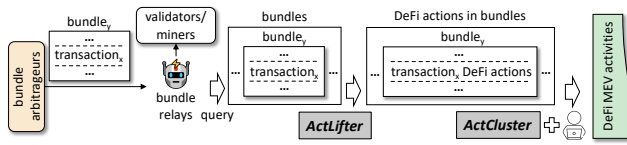
Figure 1: Workflow of discovering DeFi MEV activities in bundles. ActLifter collects the bundles constructed by bundle arbitrageurs and identifies DeFi actions in transactions of each bundle. Inspecting DeFi actions in each bundle, ActCluster facilitates us to discover DeFi MEV activities.

actions in bundles. Although DeFiRanger [89] intends to address this limitation by adopting an automated approach to recognize DeFi actions, it suffers from inaccurate recognition of DeFi actions as shown in §5.3. Second, none of them can recognize DeFi MEV activities with unknown patterns of DeFi actions.

To address the aforementioned limitations, we design a new approach shown in Fig. 1 to discover known and unknown DeFi MEV activities in bundles. We first collect bundles constructed by bundle arbitrageurs via querying Flashbots' APIs [15]. Then, to address the first limitation, we propose ActLifter (§3), a novel automated tool for accurately identifying DeFi actions in the transactions of each bundle. ActLifter first recognizes the contracts that operate the DeFi actions, the type of the DeFi actions, and the asset transfers involved in the DeFi actions according to the captured events (§3.3), then identifies DeFi actions according to the asset transfer patterns of DeFi actions (§3.4). It is worth noting that only a one-off small amount of manual effort is needed to collect events that will be emitted while executing DeFi actions, and we provide scripts to automate the process as much as possible (§3.2).

To address the second limitation, it is inevitable to involve manual inspection to uncover new DeFi MEV activities. To reduce manual efforts, we propose ActCluster (§4), a new approach that uses representation learning [67] to derive distinguishable feature vectors of bundles according to DeFi actions recognized by ActLifter, and leverages iterative clustering analysis [60] and our pruning strategies to facilitate us to discover new DeFi MEV activities.

We conduct extensive experiments (§5) to evaluate the performance of ActLifter and use ActCluster to discover DeFi MEV activities from 6,641,481 bundles (from the launch of the bundle mechanism on Feb. 11, 2021 to Dec. 1, 2022). More precisely, we evaluate the effectiveness of ActLifter in identifying ten kinds of common DeFi actions and compare it with two state-of-the-art techniques, i.e., Etherscan [2] and DeFiRanger [89]. For a fair and convincing comparison with ethical consideration, we spent more than six months in collecting 1,358,122 transactions from Etherscan to mitigate potential risks or negative effects. We queried one page of Etherscan per 10 seconds, which is slower than the human click speed, and manually solved the reCAPTCHA human authentication. The experimental results show that ActLifter outperforms existing techniques and achieves nearly 100% precision and recall. Moreover, we discovered 17 new kinds of DeFi MEV activities and three known DeFi MEV activities in bundles with the help of Act-Cluster, which reduced at least 24.2%, 97.8% and 98.8% of manual efforts than three baseline strategies.

We further demonstrate how our approach (i.e., ActLifter and ActCluster) can enhance relays' MEV countermeasures (§6.1),

Table 1: Comparison of ActLifter and other methods

Methods	SW	AL	RL	LI	NM	NB	LE	BO	AI	RE
Qin et al. [72]	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗
Qin et al. [71]	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
Wang et al. [84]	✗	✗	✗	✓	✗	✗	✓	✓	✗	✗
Wang et al. [82]	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
Mev-explore [18]	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
Piet et al. [70]	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
Weintraub et al. [85]	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗
Etherscan [2]	✓	✓	✓	✓	✓	✓	✗	✓	✗	✗
DeFiRanger [89]	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
ActLifter	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

DeFi actions. SW: Swap, AL: AddLiquidity, RL: RemoveLiquidity, LI: Liquidation, NM: NFT-Minting, NB: NFT-Burning, LE: Leverage, BO: Borrowing, AI: Airdrop, RE: Rebasing.

evaluate forking and reorganization (abbr. reorg) risks caused by bundle MEV activities (§6.2), and evaluate the impact of bundle MEV activities on blockchain users' economic security (§6.3). Moreover, we discuss three feasible usages of our approach in MEV studies, supported by our experimental results and observations (§6.4).

In summary, this work makes the following contributions.

- *First systematic analysis of DeFi MEV activities in bundles.* To our best knowledge, our work constitutes the first effort toward a systematic analysis of DeFi MEV activities conducted through Flashbots bundle mechanism after tackling two limitations.
- *Novel approach for identifying DeFi actions.* We propose ActLifter, a novel approach for automatically identifying DeFi actions from transactions, which outperforms existing techniques and achieves nearly 100% precision and recall.
- *New approach for discovering bundle MEV activities.* We propose ActCluster, a new approach facilitating us to discover bundle MEV activities with much less manual efforts. In particular, it empowers us to discover 17 new kinds of DeFi MEV activities.
- *New applications.* We demonstrate the usages of our approach (i.e., ActLifter and ActCluster), including enhancing relays' MEV countermeasures, evaluating forking and reorg risks caused by bundle MEV activities, and evaluating the impact of bundle MEV activities on blockchain users' economic security. Additionally, we discuss three feasible usages of our approach in MEV studies, supported by our experimental results and observations.

We refer readers to [29] for our full paper version with the appendix.

2 BACKGROUND AND NOTATION

This section introduces DeFi applications and actions, Flashbots bundle, and the notations used in this paper. For the basic concepts of smart contracts, events, and ERC20/ERC721 standards, we refer readers to other helpful studies [53, 63, 77]. Besides, the basic concepts of representation learning can be found in Appendix I.

2.1 DeFi Applications and Actions

We focus on ten core DeFi actions of popular DeFi applications (i.e., AMM, Lending, NFT, and Rebase Token) involved in most MEV activities [47, 71, 72, 82, 84, 89, 96–98]. Each Defi action is represented in the form $C_{DeFi}.action_{type}(params)$, where C_{DeFi}, $action_{type}$, and $params$ refer to the smart contract implementing the DeFi action, the type and the parameters of the DeFi action, respectively.

AMM. It provides functions that allow traders to perform asset exchanges over liquidity pools automatically [91]. Traders can supply or remove their assets with liquidity pools as liquidity providers,

and pay a fee to liquidity providers when they exchange assets via an AMM. We focus on three core DeFi actions supported by AMMs:

- **A1:** Swap action *AMM.Swap(x_1: Asset$_1$, x_2: Asset$_2$)*. It performs asset exchange for a trader, which lets *AMM* receive x_1 amount of *Asset$_1$* and send out x_2 amount of *Asset$_2$*.
- **A2:** AddLiquidity action *AMM.AddLiquidity(x_1: Asset$_1$, x_2: Asset$_2$, ..., x_n: Asset$_n$)* ($n > 0$). It lets *AMM* receive n assets from a liquidity provider.
- **A3:** RemoveLiquidity action *AMM.RemoveLiquidity(x_1: Asset$_1$, x_2: Asset$_2$, ..., x_n: Asset$_n$)* ($n > 0$). It lets *AMM* return n assets to a liquidity provider.

Lending. It provides loanable assets through collateralized deposits [6, 8, 9, 32, 86]. With a collateralized deposit, a borrower can take loanable crypto assets from Lendings. It uses two kinds of debt mechanisms [6, 8, 9], i.e., over-collateralization and under-collateralization, meaning that borrowers can deposit collateral assets with a higher (resp. lower) value than that of borrowed assets. We focus on three major DeFi actions supported by Lendings:

- **A4:** Borrowing action *Lending.Borrowing(x_1: Asset$_1$)*. It lets a borrower loan *Asset$_1$* from *Lending* with the over-collateral deposit [84].
- **A5:** Leverage action *Lending.Leverage(x_1: Asset$_1$)*. It lets a borrower loan *Asset$_1$* from *Lending* with the under-collateral deposit [84].
- **A6:** Liquidation action *Lending.Liquidation(x_1: Asset$_1$, x_2: Asset$_2$)*. It lets a trader send the debt *Asset$_1$* to *Lending* for repaying the debt asset and receive the collateral *Asset$_2$* from the *Lending* if the negative price fluctuation of the collateral asset happens [71].

NFT (Non-Fungible Token). It provides unique tokens to represent someone's ownership of specific crypto assets, e.g., CryptoKitties, or a physical asset, like an artwork [43]. Most NFT contracts follow the ERC721 standard [87]. We focus on two major DeFi actions supported by NFT contracts:

- **A7:** NFT-Minting action *C$_{NFT}$.NFT-Minting(tokenId$_{x_1}$: Asset$_{C_{NFT}}$)*. It lets the NFT contract *C$_{NFT}$* mint an NFT with the tokenId x_1.
- **A8:** NFT-Burning action *C$_{NFT}$.NFT-Burning(tokenId$_{x_1}$: Asset$_{C_{NFT}}$)*. It lets the NFT contract *C$_{NFT}$* burn an NFT with the tokenId x_1.

Airdrop. The airdrop is a promotional activity for bootstrapping a cryptocurrency project by spreading awareness about the cryptocurrency project [80]. A small amount of the cryptocurrency is sent to active users for free when they retweet the post sent by the project account. We focus on the following action:

- **A9:** Airdrop action *C$_{Airdrop}$.Airdrop(x_1: Asset$_1$)*. It lets the contract *C$_{Airdrop}$* send out the *Asset$_1$*.

Rebase Token. Rebase Token follows a continuous rebasing about the number of tokens in circulation (e.g., total supply in ERC20 standard) [3, 75]. For example, token holders' balances increase or decrease automatically according to the token's price evolution provided by price oracles [30]. We focus on the following action:

- **A10:** Rebasing action *C$_{Rebasing}$.Rebasing()*. It lets token holders' balances in *C$_{Rebasing}$* contract automatically increase or decrease.

2.2 Flashbots bundle

The Flashbots [16] designed the bundle mechanism in 2021. When transactions broadcast over the P2P network, bundle arbitrageurs can observe and analyze them, and include them into bundles along with other transactions. Besides, bundle arbitrageurs can adjust the order of transactions in bundles. Bundle arbitrageurs then send bundles to trusted relays privately, such as relays of Flashbots [16], Eden [13], and BloXroute [11]. The relays distribute bundles to
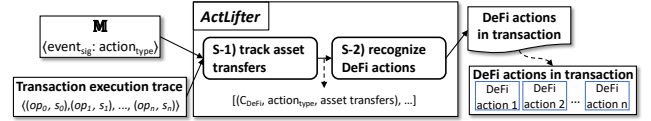

Figure 2: Overview of ActLifter.

connected miners privately. During the distribution, bundles cannot be observed by other P2P peers, until bundles are included into blocks. The connected miners will preferentially include the bundles that are the most profitable to them into the head of their mining blocks by calculating a bundle pricing formula [16].

Ethereum changed its consensus mechanism from PoW to PoS in September 2022 [22]. In the context of PoS, validators are selected to create new blocks and add blocks to the Ethereum blockchain, while miners do these tasks in the context of PoW. After the Merge, the Proposer-Builder Separation (PBS) [93] is introduced on Ethereum. In PBS, the role of validators is divided into builders and proposers. Specifically, builders create blocks with transactions from their mempool [77] and proposers submit blocks to the blockchain.

Flashbots proposed MEV-Boost [25] in 2022, which supports bundle mechanism in the context of PBS. In MEV-Boost, bundles are first propagated from arbitrageurs to builders privately. After creating blocks with bundles, builders submit blocks to relays privately with promising payments to proposers. Relays then distribute received blocks to connected proposers privately, and proposers finally pick the block with the most payments to submit to the blockchain. Currently, Flashbots [16], Eden [13], and BloXroute [11] maintain their builders and relays based on MEV-Boost. Besides, 68% of Ethereum blocks are created and relayed by MEV-Boost [25] from the starting date of MEV-Boost (Sep. 2022) to Jan. 2023 [26], and 77% of MEV-Boost blocks (i.e., blocks that are created and relayed by MEV-Boost) are from Flashbots [23]. Our studies shed light on DeFi MEV activities in bundles, since in both the context of PoW and PoS: i) arbitrageurs construct bundles, ii) bundles are relayed from arbitrageurs to validators/miners privately, and iii) validators/miners submit the most profitable bundles to them into the blockchain.

2.3 Notation

DeFi action. It, denoted as A, represents an interaction between a trader and a function provided by DeFi applications. We focus on ten kinds of DeFi actions (**A1-10** in §2.1).

DeFi actions in a transaction. A transaction can trigger the execution of multiple contracts, by invoking their functions via internal transactions. Hence, multiple DeFi actions can be operated in a transaction. We use $\mathbb{A}$ to denote a sequence of n ($n > 0$) DeFi actions operated in a transaction, where $\mathbb{A} = [A_1, A_2, ..., A_n]$. Note that these DeFi actions will be executed one by one in the order.

DeFi actions in a bundle. A bundle includes a sequence of m ($m > 0$) transactions, each of which can be signed by different accounts. These transactions will be executed one by one in the order. We use $\mathbb{B} = [\mathbb{A}_1, \mathbb{A}_2, ..., \mathbb{A}_m]$ to denote all the DeFi actions involved in a bundle.

Asset transfer. Since DeFi actions involve one or more asset transfers according to their definitions [32, 75, 80, 84, 91], to identify DeFi actions, we need to recognize asset transfers and match them against asset transfer patterns of DeFi actions. We denote an asset transfer as *Asset.Transfer(From, To, Value)*, which means *From* transfers
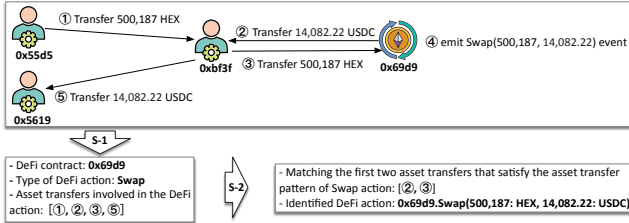
Figure 3: ACTLIFTER identifies a Swap action in a transaction.

Value amount of *Asset* to *To*, and we consider *Asset*, *From*, *To*, and *Value* are parameters of asset transfers. We consider two kinds of assets, i.e., crypto token and ETH, and distinguish them by the subscript of *Asset*. For example, $Asset_{Ether}$ refers to the ETH asset and $Asset_C$ refers to the token asset maintained by the contract *C*. We focus on ERC20 and ERC721 token assets, and our approach can be easily extended to other token assets by recognizing asset transfers from their standard events. We denote ERC721 token minting/burning as $Asset_C^{721}.Minting/Burning(From, To, Value)$, meaning that the contract *C* mints/burns an NFT of $Asset_C^{721}$ with the tokenId *Value*. With them, we can recognize the NFT-Minting and NFT-Burning actions.

Execution trace of a transaction. It refers to a sequence of states and opcodes executed in a transaction, denoted as $\langle (op_0, s_0), (op_1, s_1), ..., (op_n, s_n) \rangle$. Each state s_i is in the form of $\langle Stack_i, Memory_i \rangle$, where $Stack_i$ and $Memory_i$ are the stack [88] variables and memory [88] variables, respectively. The opcode op_i is defined in [88]. For each opcode op_i, the state s_i represents the execution environment [88] of op_i, and the state s_{i+1} denotes the state after executing op_i.

3 ACTLIFTER

3.1 Overview

As shown in Fig. 2, ACTLIFTER takes in transaction execution trace and the mapping $\mathbb{M}$ between the signature of events and the type of DeFi actions, which is constructed by a semi-automated preparation process (§3.2), and then determines DeFi actions (**A1-10** in §2.1) in the transaction by two steps.

• **S-1** (§3.3) It first locates the emitted events in the execution trace whose signatures are in $\mathbb{M}$. For each event, it outputs: i) the contract that conducts the DeFi action and emits the event, ii) the corresponding type of the DeFi action in $\mathbb{M}$, and iii) the asset transfers involved in the DeFi action.

• **S-2** (§3.4) Given the information of each event (i.e., the contract, the type of DeFi action, and the asset transfers), it recognizes corresponding DeFi action according to the asset transfer patterns (§3.4) and outputs them.

Motivating example. Fig. 3 shows how ACTLIFTER identifies a Swap action in a transaction, where there are four asset transfers (i.e., ①, ②, ③, and ⑤), and the AMM 0x69d9 emits an event Swap(*500,187, 14,082.22*) in ④.

In **S-1**, ACTLIFTER first locates the event Swap(*500,187, 14,082.22*) emitted in ④ whose signature is in $\mathbb{M}$, and then recognizes i) the contract 0x69d9 since it emits the event in ④, ii) the type of DeFi action (i.e., Swap), and iii) the four asset transfers (i.e., ①, ②, ③, and ⑤) since their parameters are logged in the event, i.e., *500,187* and *14082.22*. The outputs of **S-1** are *[(0x69d9, Swap, [①, ②, ③, ⑤])]*. In **S-2**, ACTLIFTER recognizes the Swap action $0x69d9.Swap(500,187: HEX, 14082.22:$

Algorithm 1: S-1

Input: *t*, transaction execution trace
Input: $\mathbb{M}$, mapping between the event signatures and the type of DeFi actions
Output: *[(C, action_type, assetTransfers)]*, *C* operates a DeFi action, *action_type* is the DeFi action's type, and *assetTransfers* are asset transfers involved in the DeFi action

```
1  output ← []
2  events = ParseEvents(t)
3  for event_i ∈ events do
4      if event_{i_sig} ∈ M then
5          C_i = GetContract(event_i, t)
6          action_{type_i} = GetActionType(event_{i_sig}, M)
7          assetTransfers_i = GetAssetTransfers(event_i, t)
8          output.append((C_i, action_{type_i}, assetTransfers_i))
9  return output
10 Function GetAssetTransfers(event_i, t):
11     retAssetTransfers ← []
12     assetTransfersinTrace = GetAllAssetTransfers(t)
13     for assetTransfer_j ∈ assetTransfersinTrace do
14         if IsLogged(assetTransfer_j, event_i, t) then
15             retAssetTransfers.append(assetTransfer_j)
16     return retAssetTransfers
```

USDC) in the transaction according to the asset transfer patterns of Swap action (§3.4) (i.e., the contract 0x69d9, which operates the Swap action, receives HEX in ③ and sends out USDC in ②).

DeFiRanger [89] will report a false Swap action by pairing asset transfers in ① and ⑤, because it only matches the first two asset transfers that satisfy the criteria defined in [89], i.e., one account receives and sends out different assets in two asset transfers.

3.2 Preparation

The majority of existing studies [18, 70–72, 82, 84] use specific events to recognize DeFi actions, because smart contracts use events to notify others (e.g., users, third-party tools) about their execution (e.g., state changes) [94]. Motivated by these studies, we construct a mapping $\mathbb{M}$ from the events to the corresponding type of DeFi actions by leveraging the event information from developers, which is scattered in different places, such as each DeFi's official website, document, or source codes. We first develop a tool to collect the descriptions of events or the code snippets and comments of events from the websites of popular DeFi applications listed in DeFiPulse [12] and Dapp.com [7]. Then, we manually confirm the results to construct the mapping $\mathbb{M}$.

More precisely, if a DeFi application provides documents, we summarize the document template to extract the descriptions of events in its documents. Our tool also checks whether the extracted event indeed exists in the source codes of the DeFi. If a DeFi application does not provide documents, our tool inspects its source code to extract code snippets that define events (i.e., the keyword event representing the start of an event definition, the event's name, and the definition of the event's parameters) in Solidity or Vyper, the comments of events, and functions that emit events.

Two authors read the information of extracted events independently to determine whether the events correspond to DeFi actions (Appendix A uses two examples to illustrate how we determine the results.). After analyzing the collected information, they discuss and adjust results with the help of a third author to resolve conflicts for the sake of minimizing the impacts of human subjectivity.

The whole procedure of manual analysis cost around 18 hours. We collect 32 and 56 events from the descriptions of events and the code snippets, and the comments of events, respectively. Specifically, we collect 37, 9, 12, 8, 3, 8, 7, and 4 different events for

Table 2: Conditions of asset transfers

Asset transfer type	Conditions
Ether transfer	c_1: From.CALL(To, Value) $\|\|$ TX(From, To, Value) c_2: (Value ≠ 0) ∧ (From ≠ To) $Asset_{Ether}.Transfer(From, To, Value)$
Token transfer	c_1: C.Event(Transfer(From,To,Value)) c_2: From ∉ (0x00...00, C) ∧ To ∉ (0x00...00, C) ∧ (Value ≠ 0) ∧ (From ≠ To) $Asset_C.Transfer(From, To, Value)$
ERC721 token minting	c_1: C.Event(Transfer(From,To,Value)) c_2: From ∈ (0x00...00, C) ∧ To ∉ (0x00...00, C) ∧ (C ⊨ ERC721 standard) $Asset_C^{721}.Minting(From, To, Value)$
ERC721 token burning	c_1: C.Event(Transfer(From,To,Value)) c_2: From ∉ (0x00...00, C) ∧ To ∈ (0x00...00, C) ∧ (C ⊨ ERC721 standard) $Asset_C^{721}.Burning(From, To, Value)$

Table 3: Asset transfer patterns of ten DeFi actions

DeFi action type	Asset transfer patterns
Swap	$Asset_1.Transfer(\_,C_{DeFi},x_1) ∧ Asset_2.Transfer(C_{DeFi},\_,x_2)$ $C_{DeFi}.Swap(x_1{:}Asset_1, x_2{:}Asset_2)$
AddLiquidity	$Asset_1.Transfer(\_,C_{DeFi},x_1) ∧ Asset_2.Transfer(\_,C_{DeFi},x_2) ∧ ... ∧ Asset_n.Transfer(\_,C_{DeFi},x_n)$ $C_{DeFi}.AddLiquidity(x_1{:}Asset_1, x_2{:}Asset_2, ..., x_n{:}Asset_n)$
RemoveLiquidity	$Asset_1.Transfer(C_{DeFi},\_,x_1) ∧ Asset_2.Transfer(C_{DeFi},\_,x_2) ∧ ... ∧ Asset_n.Transfer(C_{DeFi},\_,x_n)$ $C_{DeFi}.RemoveLiquidity(x_1{:}Asset_1, x_2{:}Asset_2, ..., x_n{:}Asset_n)$
Leverage	$Asset_1.Transfer(C_{DeFi},\_,x_1)$ $C_{DeFi}.Leverage(x_1{:}Asset_1)$
Borrowing	$Asset_1.Transfer(C_{DeFi},\_,x_1)$ $C_{DeFi}.Borrowing(x_1{:}Asset_1)$
Liquidation	$Asset_1.Transfer(\_,C_{DeFi},x_1) ∧ Asset_2.Transfer(C_{DeFi},\_,x_2)$ $C_{DeFi}.Liquidation(x_1{:}Asset_1, x_2{:}Asset_2)$
NFT-Minting	$Asset_{C_{DeFi}}^{721}.Minting(\_,\_,x_1)$ $C_{DeFi}.NFT\text{-}Minting(tokenId_{x_1}{:}Asset_{C_{DeFi}})$
NFT-Burning	$Asset_{C_{DeFi}}^{721}.Burning(\_,\_,x_1)$ $C_{DeFi}.NFT\text{-}Burning(tokenId_{x_1}{:}Asset_{C_{DeFi}})$
Airdrop	$Asset_1.Transfer(C_{DeFi},\_,x_1)$ $C_{DeFi}.Airdrop(x_1{:}Asset_1)$
Rebasing	$-$ $C_{DeFi}.Rebasing()$

Swap, AddLiquidity, RemoveLiquidity, Liquidation, Leverage, Borrowing, Airdrop, and Rebasing actions, respectively. Besides, we leverage the standard Transfer event in ERC20 [45] to recognize NFT-Minting and NFT-Burning actions, since the widely used contract templates for NFT (e.g., OpenZeppelin [5] and chiru-labs [4]) emit the Transfer event during NFT minting and burning.

We further investigate the events in $\mathbb{M}$ to estimate how much manual work we reduce compared to existing studies [18, 70–72, 82, 84]. To our best knowledge, previous studies conduct three steps to derive rules for recognizing DeFi actions: i) find out specific events that correspond to DeFi actions. ii) summarize how to recognize DeFi actions from the arguments of the events. iii) find out extra information (e.g., other events or storage variables) to assist in recognizing DeFi actions if they fail in the ii) step. For the 88 events in $\mathbb{M}$, we find that 41 events can be used to recognize DeFi actions according to the ii) step, and 47 events need extra manual work at the iii) step. Compared to the previous studies, we only need to find out the specific events that correspond to DeFi actions, and obviate the need of the manual efforts for the last two steps.

3.3 Step S-1

Algorithm 1 presents the process of step **S-1**. Taking in transaction execution trace and $\mathbb{M}$, AcᴛLɪꜰᴛᴇʀ first locates the emitted events whose signatures are in $\mathbb{M}$. Then, for each event, AcᴛLɪꜰᴛᴇʀ identifies and outputs the information of the corresponding DeFi action, including C_{DeFi}, $action_{type}$, and $params$ (**A1-10** in §2.1).

More precisely, AcᴛLɪꜰᴛᴇʀ locates all emitted events in the trace (Line 2) by retrieving the signature and parameters of events from the execution state (i.e., $Stack$ and $Memory$) of the opcodes used to log events, i.e., LOG0-4 [14], and only keeps the events whose signatures are in $\mathbb{M}$ (Line 3 and 4). AcᴛLɪꜰᴛᴇʀ also records the contracts that log these events in the trace [14, 38, 39, 51, 58] (Line 5) and obtains the type of the corresponding DeFi actions from $\mathbb{M}$ (Line 6).

Since $param$ of a DeFi action is summarized from asset transfers involved in the DeFi action, AcᴛLɪꜰᴛᴇʀ tracks asset transfers that are logged by the events through the function GetAssetTransfers (Line 7). These asset transfers will be used to recognize DeFi actions in **S-2**. We focus on recognizing four kinds of asset transfers described in §2.3, namely Ether transfer, token transfer, and ERC721 token minting/burning. Specifically, Ether can be transferred in two ways: i) the sender is a smart contract and executes the CALL opcode [88] by setting the recipient and the amount of transferred Ether as its parameters in the $stack$, ii) the sender is an externally-owned account (EOA) [42] and signs a transaction with setting the recipient and the amount of transferred Ether as its parameters. Moreover, if a token

transfer, or an ERC721 token minting/burning occurs, an ERC20 standard Transfer event [45] will be emitted with the parameters of the sender, the recipient, and the amount of transferred token or the tokenId of minted/burnt ERC721 token, according to the specification of ERC20 standard [45] and the widely used contract templates for ERC721 (e.g., OpenZeppelin [5] and chiru-labs [4]).

We summarize two conditions (i.e., c_1 and c_2) for identifying each kind of asset transfer. c_1 checks whether an asset transfer occurs, e.g., a sender transfers Ether to a recipient, or a Transfer event is emitted. However, asset transfers, which do not trigger the actual transfer of assets between the sender and the recipient, can pass the check of c_1 (e.g., the transferred amount of asset is zero). Thus, we use c_2 to filter out such asset transfers. Table 2 lists the four types of asset transfers and their c_1 and c_2, which are elaborated as follows. Due to the page limit, we describe how we recognize Ether transfers as follows, and introduce the rest in Appendix B.

• **Ether transfer.** In an Ether transfer $Asset_{Ether}.Transfer(From, To, Value)$, $From$ sends $Value$ amount of ETH to To. Hence, c_1 checks whether an Ether transfer occurs in any of the two cases: i) $From$ is a contract and executes a CALL to transfer $Value$ amount of Ether to To (i.e., $From.CALL(To, Value)$), ii) $From$ is an EOA account and signs a transaction to send $Value$ amount of Ether to To (i.e., $TX(From, To, Value)$). An Ether transfer should also satisfy both requirements in c_2: i) $From$ and To are different accounts (i.e., $From ≠ To$), and ii) the transferred amount $Value$ is non-zero (i.e., $Value ≠ 0$). Note that there is no actual transfer of Ether between $From$ and To if any requirement is violated.

For all asset transfers identified from the trace, we check whether they are logged by events (Line 12-14). Specifically, we check whether the event's parameters contain the asset transfer's parameters, since an event takes parameters of an asset transfer as its parameters to log the asset transfer. Asset transfers include four parameters, i.e., $Asset$, $From$, To, and $Value$. To check the first three parameters which are of the address type [27, 37, 95], we determine whether there are parameters of address type in the event, and values of the parameters are the same as the first three parameters of the asset transfer. The $Value$ parameter denotes the amount of transferred asset, and its type is a 256-bit unsigned integer [45, 88]. Since an event can convert $Value$ to another type (e.g., signed integers [27]) and use the

Algorithm 2: S-2

Input: $[(C, action_{type}, assetTransfers)]$, C operates a DeFi action, $action_{type}$ is the DeFi action's type, and $assetTransfers$ are asset transfers involved in the DeFi action
Output: $\mathbb{A}$, DeFi actions in a transaction

1   $\mathbb{A} \leftarrow []$
2   **for** $(C_i, action_{type_i}, assetTransfers_i) \in [(C, action_{type}, assetTransfers)]$ **do**
3     $A \leftarrow null$
4   **if** $action_{type_i} \in$ ("AddLiquidity", "RemoveLiquidity") **then**
5       $A = \text{GetActionOfNAssetTransfers}(C_i, action_{type_i}, assetTransfers_i)$
6   **else if** $action_{type_i} \in$ ("Swap", "Liquidation") **then**
7       $A = \text{GetActionOfTwoAssetTransfers}(C_i, action_{type_i}, assetTransfers_i)$
8   **else if** $action_{type_i} \in$ ("Rebasing") **then**
9       $A = \text{GetActionOfNullAssetTransfer}(C_i, action_{type_i}, assetTransfers_i)$
10   **else**
11      $A = \text{GetActionOfOneAssetTransfer}(C_i, action_{type_i}, assetTransfers_i)$
12   **if** $A \neq null$ **then**
13      $\mathbb{A}$.append(A)
14 **return** $\mathbb{A}$

converted one as its parameter, we also check whether absolute values of parameters in the event are the same as *Value*'s value.

3.4   Step S-2

Given the information (i.e., the contract that executes a DeFi action, the DeFi action's type, and asset transfers involved in the DeFi action) collected in **S-1**, ActLifter determines DeFi actions according to their asset transfer patterns in **S-2**. Table 3 summarizes asset transfer patterns of ten DeFi actions according to their definitions [32, 75, 80, 84, 91]. We explain them and describe how ActLifter leverages patterns to recognize DeFi actions as follows.

• **Swap.** It involves two asset transfers in the transaction, where the C_{DeFi} receives x_1 amount of asset $Asset_1$, and sends out x_2 amount of another asset $Asset_2$.

• **AddLiquidity/RemoveLiquidity.** It involves n asset transfers in the transaction. For each asset transfer, C_{DeFi} receives (resp. sends out) a different kind of asset $Asset_i$, whose amount is x_i.

• **Leverage/Borrowing.** It involves one asset transfer, where C_{DeFi} sends out x_1 amount of $Asset_1$.

• **Liquidation.** It involves two asset transfers in the transaction, where the C_{DeFi} receives x_1 amount of $Asset_1$, and sends out x_2 amount of a different asset $Asset_2$.

• **NFT-Minting/NFT-Burning.** It involves an ERC721 token minting (resp. burning), where C_{DeFi} mints (resp. burns) an NFT with the tokenId x_1.

• **Airdrop.** It involves one asset transfer, where C_{DeFi} sends out x_1 amount of $Asset_1$.

• **Rebasing.** Since no asset transfer is involved in the Rebasing action, for the contract C_{DeFi} that conducts the Rebasing action, we check whether C_{DeFi} is an ERC20 or ERC721 token contract.

Algorithm 2 presents the process of **S-2**. ActLifter takes in a list of C, $action_{type}$, and $assetTransfers$, and then recognizes the DeFi action (Line 4-11) according to the asset transfer patterns in Table 3. Finally, it outputs the recognized DeFi actions in a transaction (Line 14). Since different kinds of DeFi actions involve different numbers of asset transfers, we divide them into four categories as follows.

First, for AddLiquidity and RemoveLiquidity actions that require n asset transfers, we pick n asset transfers in a greedy fashion from aTs_i that match the patterns to recognize them (Line 6). Second, for Swap and Liquidation actions that require two asset transfers, we pick the first two asset transfers from aTs_i that match the patterns to recognize them (Line 8). Third, since Rebasing action does not
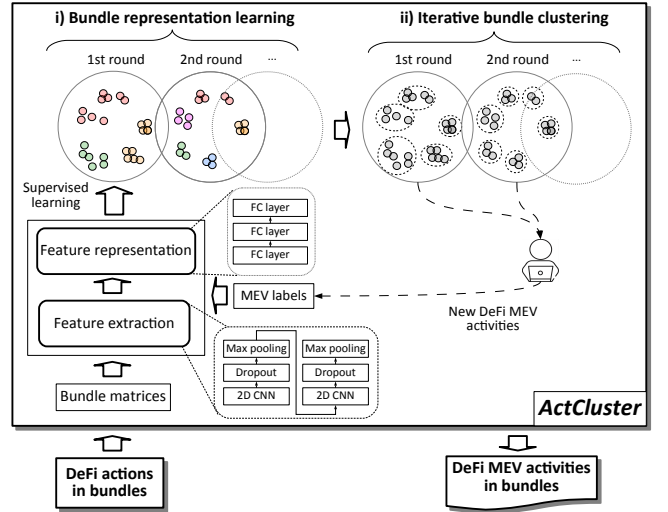

Figure 4: Overview of ActCluster

require asset transfers, for contract C_{DeFi} that conducts the Rebasing action, we check whether C_{DeFi} implements standard functions defined in ERC20 or ERC721 [48]. Fourth, for the other five DeFi actions that require one asset transfer, we pick the first asset transfer from aTs_i that matches the patterns to recognize them (Line 12).

4   ACTCLUSTER

ActCluster aims at facilitating analysts to discover DeFi MEV activities in bundles, especially the unknown ones, by analyzing the semantic features involved in the sequences of DeFi actions identified by ActLifter. As shown in Fig. 4, it consists of two steps, i.e., i) bundle representation learning (§4.1), which maps the raw bundles to their feature vectors in a low-dimensional feature space, and ii) iterative bundle clustering (§4.2), which discovers new kinds of DeFi MEV activities via iteratively clustering feature vectors of bundles. We repeat the two steps by conducting representation learning with newly discovered DeFi MEV activities in the first step and conducting the iterative clustering analysis in the second step, until we cannot find new DeFi MEV activities.

The design rationale of ActCluster is fourfold. First, manual efforts in inspecting DeFi actions in bundles are required to discover new DeFi MEV activities. We cluster bundles with similar activities to minimize the manual work. Second, there is a dilemma in the setting of clustering granularity. Specifically, bigger but sparse clusters may mix bundles containing different DeFi MEV activities together, whereas smaller but denser clusters increase manual efforts in inspecting bundles sampled from each cluster. We leverage iterative clustering analysis [60] to address the dilemma, i.e., i) it gradually improves the clustering granularity to facilitate the discovery of relatively rare DeFi MEV activities. Besides, ii) it reduces the number of clusters that need to be manually inspected through bundle pruning, which iteratively excludes bundles containing known and discovered DeFi MEV activities from the bundle dataset.

Third, conventional clustering algorithms cannot be directly applied to raw bundles due to bundles' heterogeneous format and hierarchical data structure. To tackle this problem, we employ representation learning [35] to automatically extract distinguishable

features from raw bundles with the knowledge of all known and discovered DeFi MEV activities. Unlike feature engineering, which requires rich domain-specific knowledge, representation learning is fully data-driven and task-oriented, obviating considerable manual efforts for data study. Fourth, in the first round, we conduct the representation learning with three known MEV labels collected from existing studies. Inspired by previous studies [90, 100] that improve model training's efficiency and performance by scaling up labels and iteratively training with dynamical label updating, after each round, we extend new DeFi MEV activities to MEV labels and conduct the representation learning to improve its representation capabilities for DeFi MEV activities in bundles.

4.1 Bundle representation learning

We map bundles to low-dimensional feature vectors, based on which the dissimilarity between two bundles can be quantified by the distance between their feature vectors and thus clustering analysis of bundles can be reasonably conducted.

Bundle Formatting. Raw bundles are in a heterogeneous format since a bundle contains a variable number of transactions, each of which contains a variable number of DeFi actions. To facilitate the feature extraction, we express raw bundles in a unified format, i.e., a bundle matrix with a fixed shape, because bundles in the format can be directly processed by convolutional neural network (CNN) [50] in an end-to-end fashion. Considering that bundles are organized in a hierarchical structure, we construct a bundle matrix in a bottom-up manner. Specifically, we first standardize the description of DeFi actions as ten types of parameterized action blocks, corresponding to each kind of DeFi action (**A1-10** in §2.3). As shown in Fig. 5, each DeFi action in a transaction will be expressed as an action block, acting as a basic element to describe this transaction. Sequentially concatenating action blocks corresponding to all DeFi actions within a transaction yields the transaction block that expresses this transaction. Recall that bundle is essentially a bunch of transactions. We construct the bundle matrix to express a bundle by combining transaction blocks corresponding to all transactions within it. We elaborate more on their constructions in Appendix F.

Feature extraction. We extract features from bundle matrices by taking advantage of a CNN. The reasons for choosing CNN are threefold: i) a bundle can be regarded as a time series because transactions within it are ordered. The temporal patterns involved in a bundle have been characterized as spatial patterns in our matrix representation of bundles. Thus, our task is suitable for CNN, which is known to be effective and efficient in extracting features from spatial patterns [50]. ii) typical time series analysis models are not suitable for our tasks. First, transactions cannot be represented as tokens as the input of typical models (e.g., Bert and Transformer). Second, transactions consisting of various actions and parameters are difficult to be compactly represented as feature vectors with fixed size as the input of RNN and its variants (e.g., LSTM and GRU) without information loss. iii) CNN processes data in parallel and thus is efficient, e.g., CNN-based models even achieve state-of-the-art performance in traffic analysis tasks [76], where samples are represented as time series. As shown in Fig. 4, feature extraction is implemented using stacked blocks consisting of a 2D CNN layer, a dropout layer, and a max pooling layer. Such a network structure

facilitates feature extraction because i) the 2D CNN layer with learnable kernels automatically captures informative features to construct feature maps, ii) the dropout layer reduces the overfitting risk, and iii) the max pooling layer downsamples feature maps to highlight the most important feature. The input is a bundle matrix, and the outputs are feature maps extracted by the last block.

Feature representation. To represent a bundle in a low-dimensional feature space, we flatten its feature maps obtained via feature extraction and process them with three stacked fully connected (FC) layers. The output of the last fully connected layer is the low-dimensional feature vector of this bundle. Models for feature extraction and feature representation are trained by leveraging supervised learning with the aid of all known MEV labels. Specifically, we construct a multi-label classifier based on multilayer perceptron (MLP) [74] to classify bundles in the feature space. We construct the initial MEV labels by collecting three types of MEV DeFi activities from existing studies [47, 72, 82, 98], i.e., Sandwich Attack, Cyclic Arbitrage, and Liquidation. After we discover new DeFi MEV activities in the clustering analysis of each round, we extend our MEV labels with them. MLP predicts the presence/absence of each label for a bundle. Specifically, 1 (resp. 0) indicates the presence (resp. absence) of a label. We specify the output layer of MLP as a sigmoid layer so that outputs are normalized in the range of (0, 1). We choose Mean Square Error (MSE) loss [79] to quantify the prediction error of MLP. MLP and models for feature extraction and feature representation are jointly trained by minimizing the MSE loss.

4.2 Iterative bundle clustering

Given the feature vectors of bundles, we characterize the dissimilarity between bundles with the distance between their feature vectors to facilitate the discovery of new DeFi MEV activities via bundle clustering. We test five candidate clustering algorithms, including hierarchical clustering [92], DBSCAN [92], K-means [92], Mean Shift [92], and Birch clustering [92], and finally cherry-pick DBSCAN for two reasons. First, it is more efficient than other algorithms in handling large-scale datasets in our problem. Second, DBSCAN does not need a pre-specified number of clusters.

DBSCAN is a density-based clustering algorithm. Its parameter ϵ (i.e., the maximum distance between two samples for one to be considered as the other's neighborhood [92].) adjusts the lower bound of cluster density. A larger ϵ leads to bigger but sparse clusters, where bundles corresponding to various DeFi MEV activities may be mixed together. By contrast, a smaller ϵ results in smaller but denser clusters, enabling more fine-grained clustering analysis in favor of distinguishing different DeFi MEV activities. However, a side-effect is it substantially increases manual efforts because a smaller ϵ yields more clusters and we need to manually inspect them to verify whether they contain unseen DeFi MEV activities. To address the dilemma, we leverage the iterative clustering analysis by conducting the following steps after representation learning in each round until we cannot discover new DeFi MEV activities:

- **I.** Filter out bundles that only contain DeFi actions that can make up known and discovered DeFi MEV activities.
- **II.** Group bundles into clusters based on DBSCAN.
- **III.** Discover new DeFi MEV activities by sampling one bundle from each cluster and manually inspecting them to determine whether
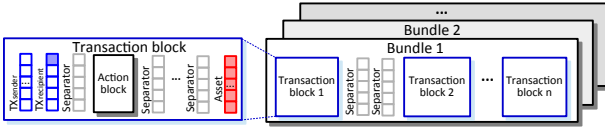
Figure 5: Bundle matrix after bundle formatting

their DeFi MEV activities are new. To avoid individual bias, we involve three authors to jointly make a decision, achieving a consensus on whether a DeFi MEV activity is new.
- **IV.** Reduce the parameter ϵ by multiplying it by a decay factor $\eta = 0.5$ to improve the resolution of clustering analysis.

Note that the number of bundles for clustering analysis decreases in iterations since bundles associated with discovered DeFi MEV activities are gradually filtered out. Besides, we reduce the parameter ϵ of DBSCAN in iterations yielding smaller but denser clusters. It enables us to conduct fine-grained clustering analysis for discovering bundles containing unknown DeFi MEV activities.

5 EVALUATION

We implement ActLifter and ActCluster in 7,832 lines of Python code, maintain an archive Ethereum node, and conduct experiments on a server with an Intel Xeon W-1290 CPU (3.2 GHz, 10 cores), and 128 GB memory to answer four research questions. **RQ1:** How is the performance of ActLifter in identifying DeFi actions? **RQ2:** Does ActLifter outperform existing techniques with respect to identifying DeFi actions? **RQ3:** How many kinds of new DeFi MEV activities does ActCluster discover? **RQ4:** Does ActCluster outperform other methods in reducing manual efforts?

5.1 Data collection

Trace collection. We invoke the `debug.traceTransaction()` [1] API of our archive Ethereum node (which is synchronized to the latest state) to get the transaction execution traces for ActLifter. **Bundle collection.** Since ActCluster needs the DeFi actions identified by ActLifter in each bundle, we collect bundles and their transactions by querying the web API [15] provided by the Flashbots [16], which displays all bundles and transactions in each bundle mined in Ethereum and relayed by Flashbots. By downloading all bundles from the starting date of bundle mechanism (i.e., Feb. 11, 2021) to Dec. 1, 2022, we collect 6,641,481 bundles and 26,740,394 transactions in total and form a dataset denoted by D_{Bundle}.

5.2 RQ1: Performance of ActLifter

In **S-1** (§3.3), after locating emitted events in $\mathbb{M}$, ActLifter recognizes asset transfers involved in DeFi actions, if the asset transfers are logged by the events. Specifically, if an event's parameters contain an asset transfer's parameters, ActLifter confirms that the asset transfer is logged by the event. There are four parameters (i.e., *Asset*, *From*, *To*, and *Value*) in each asset transfer. Hence, ActLifter can choose a different number of parameters (e.g., the *Value* parameter or all four parameters) of an asset transfer and determine whether the parameters are in the event's parameters. We evaluate ActLifter in terms of identifying DeFi actions with the following three configurations and manually determine the number of true positives (TP: a DeFi action is successfully identified), false positives (FP: a

non-DeFi action is reported by mistake), and false negatives (FN: a DeFi action is missed) due to the lack of dataset with ground-truth.

- $c1$. ActLifter chooses *Value* of each asset transfer and confirms whether the *Value* is in the event's parameters.
- $c2$. ActLifter chooses *Value* and *Asset* of each asset transfer, and confirms whether they are both in the event's parameters.
- $c3$. ActLifter chooses all four parameters of each asset transfer, and confirms whether they are all in the event's parameters.

We also compute the precision, recall, and f-score [68]. Since the number of transactions in D_{Bundle} is too large (> 10 million), we sample 1,358,122 transactions from D_{Bundle} for manual inspection and form a dataset denoted by D_{Trans}. To reduce unnecessary manual efforts and mitigate the potential negative effect of human subjectivity on determining TP/FP/FN, we first de-duplicate transactions having the same execution traces, because ActLifter will output identical results for them. Since the number of transactions after the trace-based de-duplication is still large (> 400,000), we further de-duplicate transactions having the same emitted event sequences (we will evaluate whether such de-duplication will cause errors in the following.). After the event-based de-duplication, 41,090 transactions are left for manual checking. Then, six authors manually check these 41,090 transactions. Once we get the TP/FP/FN results for a transaction, all de-duplicated transactions corresponding to this transaction have the same TP/FP/FN results.

Since manual inspection is labor-intensive and might cause errors, we conduct experiments to evaluate the quality of our TP/FP/FN results. First, we assess the performance of deduplication and provide the confidence level of our results. We randomly sample 1000 de-duplicated transactions from the 41,090 transactions, and find that all 1,000 transactions can be de-duplicated. Note that in relation to the total population (> 1 million), our sample size has a confidence interval of less than 0.27%, with 99.9% confidence. Second, we compute two statistical measures (i.e., Fleiss' Kappa [56] and Krippendorff's Alpha [57]) to assess whether our TP/FP/FN results from different authors reach a consensus. We randomly sample 500 transactions from the 41,090 transactions, and ask all six authors to report their own results. Then, we compute the Fleiss' Kappa and Krippendorff's Alpha to assess the reliability of their manual results. The results are 0.9884 and 0.9948, respectively, showing that six authors come to an almost perfect agreement.
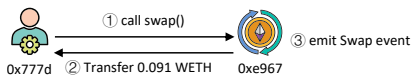
Table 4 shows the performance of ActLifter in identifying ten DeFi actions with different configurations. The third column lists the number of identified DeFi actions for each DeFi action. The fourth - ninth columns list the number of TPs, FPs, and FNs, and the values of precision, recall, and f-score for each DeFi action. It shows that ActLifter$_{c1}$ (i.e., ActLifter with the $c1$ configuration) can effectively identify DeFi actions with nearly 100% precision and recall. However, ActLifter$_{c1}$ misses 27,897 Swap actions. Manual investigation reveals that traders can receive assets from AMMs at zero cost, hence there is only one asset transfer in the transaction. Since ActLifter identifies a Swap action by matching two asset transfers (§3.4), ActLifter$_{c1}$ cannot recognize the two asset transfers in transactions, and misses identifying the 27,897 Swap actions in **S-2** (§3.4). Such cases count for only 1.26% (27,897/(27,897+2,191,810)) of all DeFi actions, and ActLifter$_{c1}$ can achieve a 98.74% recall rate. Fig. 6 shows an example, where trader 0x777d invokes swap()

Table 4: Performance metrics of ActLifter in identifying DeFi actions with different configurations

DeFi action type	Techniques	# Identified	# TP	# FP	# FN	Precision	Recall	F-score
Swap	ActLifter$_{c1}$	2,156,198	2,156,198	0	27,897	100%	98.72%	99.36%
	ActLifter$_{c2}$	102,285	102,285	0	2,081,810	100%	4.68%	8.95%
	ActLifter$_{c3}$	58,978	58,978	0	2,125,117	100%	2.70%	5.26%
AddLiquidity	ActLifter$_{c1}$	8,056	8,056	0	0	100%	100%	100%
	ActLifter$_{c2}$	45	45	0	8011	100%	0.56%	1.11%
	ActLifter$_{c3}$	45	45	0	8011	100%	0.56%	1.11%
RemoveLiquidity	ActLifter$_{c1}$	6,839	6,839	0	0	100%	100%	100%
	ActLifter$_{c2}$	1,198	1,198	0	5,641	100%	17.52%	29.81%
	ActLifter$_{c3}$	1,198	1,198	0	5,641	100%	17.52%	29.81%
Liquidation	ActLifter$_{c1}$	1,635	1,635	0	0	100%	100%	100%
	ActLifter$_{c2}$	496	496	0	1,139	100%	30.34%	46.56%
	ActLifter$_{c3}$	496	496	0	1,139	100%	30.34%	46.55%
NFT-Minting	ActLifter$_{c1}$	16,795	16,795	0	0	100%	100%	100%
	ActLifter$_{c2}$	16,795	16,795	0	0	100%	100%	100%
	ActLifter$_{c3}$	16,795	16,795	0	0	100%	100%	100%
NFT-Burning	ActLifter$_{c1}$	1,308	1,308	0	0	100%	100%	100%
	ActLifter$_{c2}$	1,308	1,308	0	0	100%	100%	100%
	ActLifter$_{c3}$	1,308	1,308	0	0	100%	100%	100%
Leverage	ActLifter$_{c1}$	34	34	0	0	100%	100%	100%
	ActLifter$_{c2}$	33	33	0	1	100%	97.06%	98.51%
	ActLifter$_{c3}$	33	33	0	1	100%	97.06%	98.51%
Borrowing	ActLifter$_{c1}$	684	684	0	0	100%	100%	100%
	ActLifter$_{c2}$	191	191	0	493	100%	27.92%	43.66%
	ActLifter$_{c3}$	191	191	0	493	100%	27.92%	43.66%
Airdrop	ActLifter$_{c1}$	246	246	0	0	100%	100%	100%
	ActLifter$_{c2}$	40	40	0	206	100%	16.26%	27.97%
	ActLifter$_{c3}$	40	40	0	206	100%	16.26%	27.97%
Rebasing	ActLifter$_{c1}$	15	15	0	0	100%	100%	100%
	ActLifter$_{c2}$	15	15	0	0	100%	100%	100%
	ActLifter$_{c3}$	15	15	0	0	100%	100%	100%
Total	ActLifter$_{c1}$	2,191,810	2,191,810	0	27,897	100%	98.74%	99.37%
	ActLifter$_{c2}$	122,406	122,406	0	2,097,301	100%	5.51%	10.45%
	ActLifter$_{c3}$	79,099	79,099	0	2,140,608	100%	3.56%	6.88%

Table 5: Performance metrics of Etherscan, DeFiRanger, and EventLifter in identifying DeFi actions

DeFi action type	Techniques	# Identified	# TP	# FP	# FN	Precision	Recall	F-score
Swap	Etherscan	1,983,869	1,983,869	0	200,226	100%	90.83%	95.20%
	DeFiRanger	1,760,236	1,356,586	403,650	827,509	77.07%	62.11%	68.79%
	EventLifter	102,285	102,285	0	2,081,810	100%	4.68%	9.16%
AddLiquidity	Etherscan	4,964	4,964	0	3,092	100%	61.62%	76.25%
	DeFiRanger	24,234	4,116	20,118	7,237	16.98%	36.25%	23.13%
	EventLifter	45	45	0	8,011	100%	0.56%	1.11%
RemoveLiquidity	Etherscan	2,629	2,629	0	4,210	100%	38.44%	55.53%
	DeFiRanger	12,289	1,143	11,146	8,270	9.3%	12.14%	10.53%
	EventLifter	1,198	1,198	0	5,641	100%	17.52%	29.81%
Liquidation	Etherscan	527	527	0	1,108	100%	32.23%	48.75%
	DeFiRanger	-	-	-	-	-	-	-
	EventLifter	496	496	0	1,139	100%	30.34%	46.55%
NFT-Minting	Etherscan	12,532	12,532	0	4,263	100%	74.62%	85.46%
	DeFiRanger	-	-	-	-	-	-	-
	EventLifter	-	-	-	-	-	-	-
NFT-Burning	Etherscan	1,190	1,190	0	118	100%	90.98%	95.28%
	DeFiRanger	-	-	-	-	-	-	-
	EventLifter	-	-	-	-	-	-	-
Leverage	Etherscan	-	-	-	-	-	-	-
	DeFiRanger	-	-	-	-	-	-	-
	EventLifter	33	33	0	1	100%	97.06%	98.51%
Borrowing	Etherscan	141	141	0	543	100%	20.61%	34.18%
	DeFiRanger	-	-	-	-	-	-	-
	EventLifter	191	191	0	493	100%	27.92%	43.66%
Airdrop	Etherscan	-	-	-	-	-	-	-
	DeFiRanger	-	-	-	-	-	-	-
	EventLifter	40	40	0	206	100%	16.26%	27.97%
Rebasing	Etherscan	-	-	-	-	-	-	-
	DeFiRanger	-	-	-	-	-	-	-
	EventLifter	15	15	0	0	100%	100%	100%
Total	Etherscan	2,005,852	2,005,852	0	213,560	100%	90.38%	94.95%
	DeFiRanger	1,796,759	1,361,845	434,914	843,016	75.79%	61.77%	68.06%
	EventLifter	104,303	104,303	0	2,097,301	100%	4.74%	9.05%

of AMM 0xe967 (i.e., an AMM which supports asset exchanges between MCC and WETH tokens) in ① in a transaction. Then AMM 0xe967 is aware of a difference of MCC between AMM 0xe967's token balance and AMM 0xe967's reserve variables of MCC. Please note that AMM 0xe967's reserve variables of MCC are stored in AMM 0xe967's contract with aiming of recording AMM 0xe967's token balance of MCC. AMM 0xe967 considers that the difference of MCC is transferred by trader 0x777d, and trader 0x777d aims to buy WETH. Hence, AMM 0xe967 transfers WETH to trader 0x777d in ② and emits a Swap event in ③. Since there is only one asset transfer in the transaction, ActLifter$_{c1}$ considers there is no Swap action.

Unfortunately, ActLifter$_{c2}$ and ActLifter$_{c3}$ can only achieve 5.51% and 3.56% recall rates. Manual investigation reveals two reasons for FNs. First, for the 27,897 Swap actions missed by ActLifter$_{c1}$, both ActLifter$_{c2}$ and ActLifter$_{c3}$ also missed the 27,897 Swap actions by the same reason. Second, ActLifter$_{c2}$ and ActLifter$_{c3}$ missed recognizing 2,069,419 and 2,112,711 DeFi actions, because, due to the configurations of $c2$ and $c3$, the asset transfers for identifying DeFi actions are filtered out. For the example in Fig. 3, ActLifter$_{c2}$ and ActLifter$_{c3}$ filter out the two asset transfers in ② and ③, because only the *Value* parameters of the two asset transfers (i.e., *14,082.22* and *500,187*) are logged by the Swap(*500,187, 14,082.22*) event in ④. Hence, ActLifter$_{c2}$ and ActLifter$_{c3}$ can not identify the corresponding DeFi action, which is matched by the two asset transfers in ② and ③. By contrast, ActLifter$_{c1}$ can recognize asset transfers involved in DeFi actions (e.g., the two asset transfers in ② and ③), and hence it can identify the corresponding DeFi actions. Since ActLifter with the $c1$ configuration achieves nearly 100% accuracy and significantly outperforms ActLifter under the

Figure 6: An example of ActLifter$_{c1}$'s false negatives

other two configurations, i.e, ActLifter$_{c2}$ and ActLifter$_{c3}$, we run ActLifter with the $c1$ configuration for other experiments.

Insight. When DeFi developers emit events to announce DeFi actions, we find that most of them only publish the amount of transferred assets involved in DeFi actions without other parameters of asset transfers (e.g., the type of asset). It may lead to potential risks for traders in interacting with DeFis, because the same events may be triggered by DeFi actions involving different kinds of assets and thus traders will get confused or abused by adversaries.

Answer to RQ1: *ActLifter can achieve nearly 100% precision and recall in identifying ten kinds of DeFi actions.*

5.3 RQ2: Is ActLifter superior to others?

We compare ActLifter with three baseline methods, including two state-of-the-art techniques (i.e., Etherscan [2] and DeFiRanger [89]), and EventLifter, a tool we developed for recognizing DeFi actions from events' arguments, because, as mentioned in §3.2, we find 41 events whose arguments can be leveraged to recognize DeFi actions. We compare their performance in terms of identifying DeFi actions for transactions in D_{Trans}. Since Etherscan does not release DeFi action results in its APIs, we queried their transaction pages to obtain DeFi action results. Since DeFiRanger is also not available [89], we re-implemented its DeFi action identification approach. Note that DeFiRanger [89] only identifies AddLiquidity and RemoveLiquidity actions that supply and withdraw single asset with AMMs. For example, if an AddLiquidity action supplies two assets $Asset_1$ and $Asset_2$ to an AMM, DeFiRanger will identify two AddLiquidity actions that supply $Asset_1$ and $Asset_2$ to the AMM, respectively. We still consider that DeFiRanger identifies the true results, if their results can be combined into the true DeFi actions.

Table 5 shows the results of Etherscan, DeFiRanger, and EventLifter. The third - ninth columns list the number of identified DeFi actions, TPs, FPs, and FNs, and the values of precision, recall, and f-score

for different kinds of DeFi actions. We next present reasons why three baseline techniques generate FP and FN cases.

Etherscan. Etherscan achieved 100% precision for the identification of 7 kinds of DeFi actions but generated incomplete results. For example, Etherscan missed identifying a Swap action in the transaction [21], where the trader exchanges `COMP` for `WETH` with the AMM `0xba12`. Both ActLifter and DeFiRanger can correctly identify this Swap action. Since Etherscan does not disclose how they identify DeFi actions, we cannot know why Etherscan failed.

DeFiRanger. DeFiRanger led to both incomplete and inaccurate results in identifying three kinds of actions. The root causes are twofold: i) DeFiRanger identifies DeFi actions by matching ERC20 token transfers [89], and thus it cannot identify DeFi actions involving Ether transfers. ii) Heuristics, defined by DeFiRanger [89], are inaccurate. For example, DeFiRanger identifies Swap actions by only matching the first two token transfers, and in the two matched token transfers one account receives and sends out different assets. However, token transfers, which are irrelevant to DeFi actions, can also satisfy these heuristics. Hence, DeFiRanger will wrongly match the irrelevant token transfers and report wrong DeFi actions. For the example in Fig. 3, DeFiRanger identifies an incorrect Swap action by matching two irrelevant token transfers in ① and ⑤.

EventLifter. EventLifter only achieved 4.74% recall rate and missed identifying 2,097,316 DeFi actions. It shows that the 41 kinds of events only count for a small proportion (i.e., 4.74%) of all emitted events whose signatures are in $\mathbb{M}$.

Answer to RQ2: *ActLifter can significantly outperform the state-of-the-art techniques, other baseline methods, and two variants of ActLifter in identifying DeFi actions.*

5.4 RQ3: DeFi MEV activities discovery

The representation learning of ActCluster leverages three initial MEV labels, i.e., Sandwich Attack, Cyclic Arbitrage, and Liquidation, to map each bundle matrix to the low-dimensional feature space in the first round (§4.1). We generate the initial MEV labels for each bundle in D_{Bundle} by using heuristics from Qin et al. [72]. As a result, 813,188, 1,334,207, and 14,263 bundles are labeled as Sandwich Attack, Cyclic Arbitrage, and Liquidation, respectively. Besides, two parameters are used in iterative bundle clustering (§4.2) of ActCluster, i.e., ϵ, which is used to adjust the lower bound of cluster density in DBSCAN [92], and η, which is the delay factor of ϵ. The values of ϵ and η are selected by grid search [81] with the target of minimizing required manual efforts (i.e., the amount of bundles manually inspected) to discover MEV activities. Specifically, we first make a set of candidate values for ϵ and η, and then perform the iterative bundle clustering (§4.2) with each pair of parameters on a small set of (i.e., 5,000) bundles in D_{Bundle}. Finally, we compare the amount of bundles manually inspected in iterative bundle clustering (§4.2), and determine 16 and 0.5 for ϵ and η, respectively.

We train on all our data (i.e., 6,641,481 bundles in D_{Bundle}) with MEV labels. After each round in ActCluster, we add the newly discovered MEV activities into the MEV labels, and conduct the representation learning (§4.1) of the next round with the extended MEV labels. After repeating the steps of ActCluster (§4) four rounds and analyzing 2,035 bundles manually, we discover 17 new
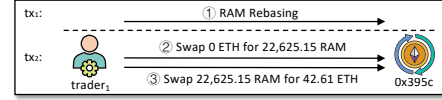

Figure 7: An example of Rebasing Backrun Arbitrage

kinds of DeFi MEV activities summarized in Table 6. We describe one as follows, and introduce the rest in Appendix E.

• **Rebasing Backrun Arbitrage (RBA).** It involves two transactions in a bundle. The former executes a Rebasing action, which causes a difference between the AMM's Rebase token balance and the AMM's reserve variables. The Rebase token balance is stored in the contract of Rebase token, and the reserve variables are stored in the contract of AMM with recording the AMM's Rebase token balance. The latter executes a Swap action to trade the Rebase token and obtains profits from the difference of Rebase token. For example, Fig. 7 shows the two transactions in the first bundle of the 12,147,015 block. In the first transaction, the `RAM` token executes a Rebasing action, and it causes the `RAM` token balance of the AMM `0x395c` to increase by 22,625.15. However, the AMM `0x395c` still uses its old `RAM` token balance (i.e., the reserve variables) before the Rebasing action to calculate how much the traders should pay [30]. In the second transaction, after giving the trader the 22,625.15 `RAM`, the AMM `0x395c` finds its `RAM` token balance does not decrease. Hence, the $trader_1$ does not need to pay for `ETH`. The $trader_1$ then swaps the 22,625.15 `RAM` for 42.61 `ETH` to earn profits of 42.61 `ETH`.

We also evaluate whether our results can generalize to new MEV activities. We first train our model on a small set of (i.e., 20,000) bundles in D_{Bundle}, and then evaluate our trained model on different validation sets [67] (i.e., other bundle sets randomly sampled from D_{Bundle}). It shows that our trained model achieves similar accuracy (difference < 5%) in classifying MEV labels on different validation sets. It means that our model could generalize to different sets of bundles. As new DeFi MEV activities can cause concept drift [49] in bundles (an open problem in machine learning) and might affect the accuracy of our model, Users can retrain [49] our model with new MEV activities. We evaluated the retraining cost of our model, and the result shows that it is reasonable (Appendix J).

Answer to RQ3: *ActCluster empowers us to discover 17 new kinds of DeFi MEV activities in bundles. Besides, our results can generalize to new types of DeFi MEV activities.*

5.5 RQ4: Is ActCluster superior to others?

To evaluate how much manual effort can be reduced by ActCluster. We compare it with three baseline strategies. It is worth noting that the three baseline strategies are selected with ablating components of ActCluster. Hence, by comparing with the three baseline strategies, we can also evaluate to which extent the components of ActCluster benefit the procedure of MEV activity discovery.

• **ActCluster⁻.** It ablates the updating of labels of newly discovered MEV activities in model training of bundle representation learning (§4.1). We only conduct the representation learning (§4.1) with the initial three MEV labels (§4.1), and conduct the iterative clustering analysis by repeating the same four steps in §4.2, until we find all 17 DeFi MEV activities.

• **Intuitive clustering analysis.** It ablates both the iterative bundle clustering (§4.2), and the updating of labels of newly discovered

Table 6: Descriptions for the 17 kinds of new DeFi MEV activities in bundles

DeFi MEV activity	Description
Swap Backrun Arbitrage	On the same AMM, the arbitrageur just backruns another trader's Swap action by a Swap action, and earns profits from the pulled-up price.
Liquidity Backrun Arbitrage	On the same AMM, the arbitrageur backruns another trader's AddLiquidity/RemoveLiquidity action by a Swap action, and earns profits from the pulled-up price.
Liquidity Sandwich Arbitrage	On the same AMM, the arbitrageur frontruns and backruns another trader's Swap action by AddLiquidity and RemoveLiquidity actions, and earns profits from the trader's exchange fee.
Multi-layered Burger Arbitrage	On the same AMM, the arbitrageur frontruns and backruns other traders' Swap actions by Swap actions, and earns profits from the pulled-up price.
Liquidity-swap Trade	On the same AMM, the trader both performs a Swap action, and performs the AddLiquidity or RemoveLiquidity actions. The trader aims to supply, withdraw, or trade assets at the expected prices.
Partial Cyclic Arbitrage	The arbitrageur performs Swap actions among AMMs to earn profits from the unbalanced prices, and part of the Swap actions can form a loop one by one.
Backrun Cyclic Arbitrage	The arbitrageur backruns another trader's Swap/AddLiquidity/RemoveLiquidity action, and performs Cyclic Arbitrage to earn profits from the unbalanced prices.
Hybrid Arbitrage	There are at least two kinds of MEV activities of known MEV activities in a bundle. There exists a transaction contained in all these MEV activities.
Failed Arbitrage	The arbitrageur suffers the financial loss, when the arbitrageur aims to obtain profits by Sandwich Attack or Cyclic Arbitrage activities.
Non-cyclic Swap Trade	The trader only performs the non-cyclic Swap actions, and aims to trade on the AMMs at the expected prices.
Rebasing Backrun Arbitrage	The arbitrageur backruns a Rebasing action by a Swap action, and earns profits from the price differences of the Rebase token.
Airdrop-swap Trade	The trader first claims the airdrop rewards, then sells the received rewards to an AMM by a Swap action.
Bulk NFT-Minting	The NFT contract mints NFTs in bulk, and it aims to increase the maintained NFTs at the expected blockchain state.
NFT Reforging	The NFT contract reforges an NFT to update the asset represented by the NFT.
Airdrop Claiming	The trader only claims and receives airdrop rewards.
NFT-Minting-swap Trade	The trader first receives an NFT minted by NFT contract, then sells the minted NFT to an AMM by a Swap action.
Loan-powered Arbitrage	The arbitrageur loans assets from Lending under the over/under-collateral deposit, then uses the loaned assets to conduct MEV activities, e.g., Cyclic Arbitrage.

MEV activities in model training of bundle representation learning (§4.1). We apply the DBSCAN algorithm to all bundles in D_{Bundle} once to find different kinds of DeFi MEV activities. Then, we sample one bundle from each cluster, and determine whether it contains new DeFi MEV activities. Since we aim to compare ActCluster with the best performance of the intuitive clustering analysis, we adjust the ϵ parameter of the DBSCAN clustering algorithm to find all 17 kinds of DeFi MEV activities.

• **Random sampling analysis.** It ablates the whole process of ActCluster. We sample one bundle from D_{Bundle} randomly, and determine whether it contains discovered DeFi MEV activities. If that is the case, we exclude all bundles containing the corresponding DeFi MEV activities from D_{Bundle}. Note that the excluded bundles only contain DeFi actions that can form the corresponding DeFi MEV activities. We repeat the random sampling analysis until we find all 17 kinds of new DeFi MEV activities.

For each strategy, we record the number of bundles to be inspected for discovering all 17 kinds of new DeFi MEV activities. Our experimental results show that ActCluster⁻, intuitive clustering analysis, and random sampling analysis, require us to manually analyze 2,874, 108,962, and 176,255 bundles, respectively. Compared to them, ActCluster can reduce 29.2%, 98.1% and 98.8% of manual efforts for discovering DeFi MEV activities, respectively.

Answer to RQ4: *ActCluster outperforms three baseline strategies in reducing manual efforts during discovering DeFi MEV activities.*

6 APPLICATIONS OF OUR APPROACH

We use three applications to demonstrate usages of our approach (i.e., ActLifter and ActCluster), including enhancing relays' MEV countermeasures (§6.1), evaluating forking and reorg risks caused by MEV activities in bundles (§6.2), and evaluating the impact of MEV activities in bundles on blockchain users' economic security (§6.3). Moreover, we discuss three feasible usages of our approach, supported by experimental results and observations (§6.4).

6.1 Enhancing MEV countermeasures in relays

As the most popular platforms implementing MEV countermeasures in practice [93], relays that distribute bundles to miners/validators can filter out bundles including known MEV activities [24]

(e.g., relays [11, 28] block sandwich attacks). However, these relays [11, 28] rely on handcrafted heuristics [19] to detect and filter out the bundles containing known MEV activities. Hence, these relays can fail to counter bundles only containing unknown MEV activities because these bundles can fail heuristics of these relays. We develop a tool named MEVHunter based on our approach to enhance relays to counter bundles containing new MEV activities. Specifically, MEVHunter takes in a bundle of transactions as input, and identifies the kinds of MEV activities (including known and our newly discovered MEV activities) exist in the bundle. For each transaction in the bundle, MEVHunter utilizes ActLifter to recognize DeFi actions in it. Besides, for each kind of MEV activity discovered by ActCluster, we summarize heuristics to identify it like others [47, 72, 85]. For each kind of MEV activity, the heuristics describe DeFi actions that a bundle arbitrageur has to perform to accomplish the corresponding MEV activity. By checking whether the DeFi actions in the bundle satisfy our summarized heuristics, MEVHunter identifies MEV activities in the bundle.

To evaluate how MEVHunter enhance MEV countermeasures in relays, we use it to inspect MEV activities for all bundles in D_{Bundle}. The experimental results show that 31.81% (2,112,344/6,641,481) bundles contain known MEV activities (i.e., Sandwich Attach, Cyclic Arbitrage, and Liquidation), and 53.12% (3,527,655/6,641,481) bundles contain our newly discovered DeFi MEV activities. Among the 3,527,655 bundles, 3,182,363 bundles only contain new DeFi MEV activities. The experimental results indicate that, MEVHunter can enhance relays to additionally identify 3,182,363 (47.92%) bundles only containing the 17 kinds of new MEV activities. We further investigate new MEV activities in bundles, e.g., the number of contracts directly invoked by the EOA account to perform new MEV activities. Our empirical results show that new MEV activities are commonly used in bundles (cf. Appendix G for details).

Summary: *Our approach can enhance MEV countermeasures in relays to discover more MEV activities in bundles, and filter out more bundles (relayed by them) containing MEV activities.*

6.2 Evaluating forking and reorg risks caused by bundle MEV activities

Prior studies [40, 61, 72, 96] report that financially rational miners are incentivized to deliberately fork and reorganize the blockchain

to gather revenues from MEV activities. Hence, we evaluate forking and reorg risks in blockchain consensus security caused by known and new MEV activities in bundles (i.e., bundle MEV activities) by measuring how many revenues miners can gather from bundle MEV activities. Our methodology for determining miners' revenues from known and new MEV activities in bundles in D_{Bundle} involves two steps. First, we recognize bundles containing known and new MEV activities by using MEVHunter as discussed in §6.1. Second, following methods in [61], we determine miners' revenues from bundle MEV activities by using miners' revenues from corresponding bundles. Miners' revenues from bundles consist of two parts: i) gas fees for transactions in bundles, and ii) Ether transfers to miners in bundles (both of them are publicly available through the web API [15]). To facilitate analysis, we combine miners' revenues from bundle MEV activities per block, and form a dataset denoted by $D_{Revenue}$, because miners' revenues from bundle MEV activities contained in the same block will cumulatively incentivize miners to fork and reorganize the blockchain. As a result, miners receive revenues from bundle MEV activities in 1,791,891 blocks. In block 14,953,916, miners received the highest revenues from bundle MEV activities as 1,584.4 Ether (792.2 times the block reward).

To further investigate how bundle MEV activities incentivize miners to fork and reorganize the blockchain, by adapting the MDP framework [72], we quantify the minimum mining power of miners incentivized to fork and reorganize the blockchain for gathering revenues in $D_{Revenue}$. Specifically, the MDP framework employs a Markov Decision Process [96] for miners to identify whether to fork and reorganize the blockchain or not, with a given mining power on various revenues. The results are shown in Fig. 8, where each point (x, y) in the red line indicates that, miners' revenues from bundle MEV activities in a block (which are x times the block reward) will incentive miners with no less than y mining power to fork and reorganize the blockchain for gathering the revenues. Besides, in Fig. 8, we display the distribution of miners' revenues from bundle MEV activities in blocks in $D_{Revenue}$ with binning in twelve intervals. Fig. 8 shows that 1,403 blocks incentivize miners with no less than 10% mining power to fork and reorganize the blockchain. Moreover, the miners' revenues from bundle MEV activities in block 14,953,916 can incentivize a miner with only 0.06% mining power to fork and reorganize the blockchain, highlighting the severe of forking and reorg risks caused by MEV activities in bundles.

Ethereum changed its consensus mechanism from PoW to PoS in September 2022 [22], and the new PoS consensus mechanism is under the forking and reorg risks undertaken by validators [41]. Besides, several studies [65, 66] propose various attacks to decrease the cost for launching forking and reorg for Ethereum blockchain. Considering that validators collect the same revenues from bundle MEV activities as miners [16, 25], we believe that bundle MEV activities still endanger the consensus security in the context of PoS by incentivizing validators to fork and reorganize the blockchain.

Summary: *Bundle MEV activities endanger the consensus security by incentivizing miners/validators to fork and reorganize the blockchain for gathering revenues from bundle MEV activities.*
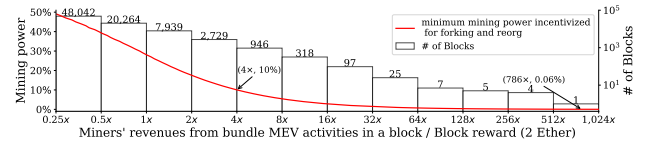

Figure 8: The minimum mining power of a miner incentivized for forking and reorg to gather miners' revenues from a block, and the distribution of miners' revenues in blocks.

6.3 Evaluating impact of bundle MEV activities on blockchain users' economic security

To explore the impact of bundle MEV activities on blockchain users' economic security, we use the Granger causality test [46, 78] to examine the range of later blocks in which users' transactions are delayed due to bundle MEV activities in prior blocks. In the context of PoW, delayed waiting time is one of the major economic security issues for users caused by MEV activities [61, 72]. For instance, it prolongs users' transactions to be exposed to arbitrageurs, thereby enhancing arbitrageurs to design and engage in more profitable MEV activities (e.g., Sandwich Attack and Cyclic Arbitrage). The Granger causality test is a statistical hypothesis test to determine whether the changes in one time series cause changes in another time series, and it is widely employed in the fields of economics, political science, and epidemiology [46, 78]. We capture the two parts of data examined for our Granger causality test in the following.

Transaction waiting times. We define the waiting time of a transaction as the duration that the transaction remains in mempools of miners/validators before being submitted to blockchain. To capture transaction waiting times, we utilize the three-month waiting time dataset (from Jul. 20, 2021 to Oct. 27, 2021) released by [61]. Additionally, we obtained a nine-day waiting time dataset for transactions from Mar. 14, 2023 to Mar. 22, 2023 by implementing the same methods as [61] (cf. Appendix K for details). We use median values of transaction waiting times in each block to account for the variation of transaction waiting times in blocks, which is more tolerant of outliers than the mean and standard deviation [61]. Finally, we combine waiting times from two time periods to form a new dataset denoted by $D_{Waiting}$, which includes the 25th, 50th, and 75th quartiles of waiting times per block (where the 25th, 50th, and 75th quartiles of waiting times are sorted in ascending order).

Extractable value. Following the methods in [61], we estimate the extractable value of bundle MEV activities by using revenues of miners/validators from bundle MEV activities (§6.2). It benefits us in estimating the extractable value of bundle MEV activities even if assets in MEV activities do not have price information for calculating the extractable value [61, 72]. Please note that $D_{Waiting}$ includes waiting times for two periods. For the first period (i.e., from Jul. 20, 2021 to Oct. 27, 2021), we obtain the extractable value of bundle MEV activities in blocks by using corresponding results in $D_{Revenue}$ (§6.2). Moreover, to obtain the extractable value of bundle MEV activities in blocks for the second period (i.e., from Mar. 14, 2023 to Mar. 22, 2023), we first capture bundles from the web API [15], and then use the methods in §6.2 to obtain the extractable value of bundle MEV activities in blocks. Finally, we combine two parts of results to form a new dataset denoted by D_{Value}.
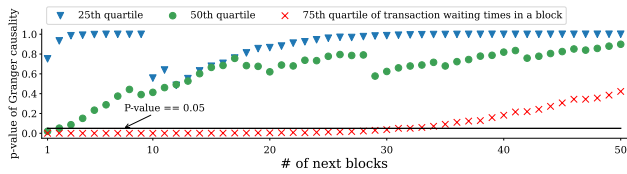
Figure 9: Bundle MEV activities in blocks cause the delayed transaction waiting times in the next x-th block, if the corresponding p-value is smaller than 0.05.

Our Granger causality test serves to examine the range of later blocks in which the waiting times of users' transactions are prolonged by prior bundle MEV activities. Inspired by the lagged Granger causality analysis [46], we achieve this purpose by lagging x block ($1 \leq x \leq 50$) of data in D_{Value}, and conducting the Granger causality test for the data in $D_{Waiting}$ and x-lagged data in D_{Value}. If the p-value of corresponding Granger causality test is smaller than 0.05, we confirm that bundle MEV activities cause the increase of corresponding transaction waiting times in the later x-th block (at the 5% level of significance [46, 78]). Our results are illustrated in Fig. 9. It shows that bundle MEV activities cause the increase of transaction waiting times at the 25th, 50th, and 75th quartiles in next 0, 2, and 30 blocks, respectively. Hence, it indicates that bundle MEV activities in blocks cause delayed waiting times of transactions in later blocks. Please note that, since miners/validators prioritize transactions with higher fees [61, 69], transactions with lower fees are commonly positioned in the back of the block [61, 69]. Hence, our results also indicate that the further back in the block a transaction will be positioned (i.e., the transaction has a lower fee), the more continuous delay that bundle MEV activities cause on its waiting time. We validate our results by measuring the correlation between bundle MEV activities and transaction waiting times via correlation tests (e.g., Spearman [64]). Our results show that Spearman coefficients between bundle MEV activities in blocks in D_{Value} and transaction waiting times at the 25th, 50th, and 75th quartiles in blocks in $D_{Waiting}$ are 0.230, 0.233, and 0.214, respectively. It indicates that as the extractable value of bundle MEV activities increases, transaction waiting times in blocks correspondingly increase [64]. Hence, the results provide further evidence of bundle MEV activities on delaying transaction waiting times.

Summary: *Bundle MEV activities endanger blockchain users' economic security by delaying users' transactions in later blocks.*

6.4 Other feasible applications

• Our approach can automatically uncover the evolving strategies of arbitrageurs (e.g., [33]) for extracting MEV from new DeFi applications in practice. As a result, our approach uncovered 17 new MEV strategies on five kinds of popular DeFi applications (i.e., AMM, Lending, NFT, Airdrop, and Rebase Token) in §5.4. Compared to us, existing studies [31, 33, 96, 98] only manually design strategies for extracting MEV from a few DeFi applications that are well-studied (e.g., AMM). The new strategies discovered by our approach can motivate researchers to further explore the design space of MEV strategies. We use [31] as an example to illustrate how our uncovered MEV strategies can facilitate relevant studies [31, 33, 96, 98] to design new strategies to extract MEV from these applications. Specifically, in Rebase Backrun Arbitrage (RBA in §5.4), we found

that an arbitrageur can backrun a Rebase token to steal assets in AMMs with zero cost. Please note that [31] explores possible MEV activities on AMMs by only manually modeling AMMs. Our findings on RBA can help [31] design new MEV strategies for AMMs (e.g., RBA) by including the modeling of the Rebase token.
• Considering the fast-growing amount of bundles (e.g., there are more than 6,641,481 bundles and 26,740,394 transactions in bundles until Dec. 2022), our approach can be used to detect and discover MEV activities in bundles continuously. As a result, by automatically analyzing 6,641,481 bundles, our approach detected 2,112,344 bundles containing known MEV activities, and 3,182,363 bundles only containing new MEV activities (§6.1). Compared to us, existing work [40, 47, 61, 72] on quantifying MEV activities involves lots of manual efforts. Our approach can benefit relevant studies [40, 47, 61, 72] through using our approach and its results. For instance, [72] can leverage our approach to detect MEV activities, and then leverage its results to conduct an in-depth study (e.g., [72] investigated arbitrageurs' strategies in MEV activities).
• Our approach can recognize stealthy attacks. Stealthy attacks are launched by attackers via exploiting bundles. Without bundles, attackers have to broadcast their transactions in the P2P network, and their transactions may also be attacked by other attackers. Our approach recognized three stealthy attacks in D_{Bundle} (cf. details in Appendix H) through ACTCLUSTER when we leveraged ACTCLUSTER to discover DeFi MEV activities (§5.4). Although Zhou et al. [99] reported stealthy attacks, our approach distinguishes them in three points: i) different from Zhou et al. [99] that collected attacks from literature and confirmed stealthy ones manually, our approach can automatically recognize stealthy attacks through ACTCLUSTER by recognizing outliers; ii) Given the fast-growing amount of bundles, the emerging new DeFi applications, and the evolving MEV strategies, our approach empowers the continuous detection and recognition of stealthy attacks, whereas Zhou et al. [99] only collected stealthy attacks from literature; iii) Our approach categorizes stealthy attacks by clustering them in ACTCLUSTER, whereas Zhou et al. [99] manually categorized two types of stealthy attacks.

7 THREATS TO VALIDITY

Due to the lack of ground-truth dataset, we manually analyze DeFi actions identified by ACTLIFTER and baseline techniques (e.g., Etherscan and DeFiRanger). Since manual inspection is labor-intensive, we did not check whether there is any DeFi action missed by all techniques, and thus the result of false negatives might be affected. In the future, we will involve more efforts to inspect all 6,641,481 bundles to detect FNs missed by all techniques.

It raises threats to validity that we do not analyze all relays' bundles in blockchains, since different relays on a chain and relays on different chains have different strategies for relaying bundles (e.g., whether to follow censorship [24]). However, these strategies do not change how bundle arbitrageurs perform MEV activities (e.g., manipulating transactions' positions). Hence, our approach can be generalized to bundle MEV studies in the wider ecosystem. Moreover, we have analyzed multiple relays' bundles. It is worth noting that, for relays disclosing bundles relayed by them, Flashbots will collect their bundles and list them in Flashbots' web API [15]. Hence, the bundles collected in §5.1 also contain bundles relayed

by other disclosed bundle relays, e.g., Eden [13]. In future work, we will investigate relays that do not disclose their relayed bundles.

The completeness for representing DeFi actions in bundles into the low-dimensional feature space (§4) and labeling bundles with MEV activities in the feature space (§4) cannot be provably guaranteed due to the lack of ground truth. Thus, MEV activities involving other DeFi actions (that are not in **A1-10** in §2.1) can be missed as false negatives. Although our ten manually selected DeFi actions (which are heavily involved in MEV activities) are by no means complete, our approach can be easily extended to discover more new DeFi MEV activities by including more kinds of DeFi actions. In future work, we will inspect more kinds of DeFi actions.

The delayed transactions can result from various factors such as MEV activities, P2P network congestion [61], and gas fee volatility of transactions [61]. Our Granger causality test (§6.3) determines that bundle MEV activities can contribute to the delay of users' transactions. In future work, we will explore to what extent these factors contribute to increase delays of users' transaction.

While collecting DeFi actions from Etherscan (§5.3), we have taken ethical considerations by limiting our collection of DeFi actions to a slow pace (i.e., querying one page per ten seconds) and manually solving the reCAPTCHA human authentication. However, our collection of DeFi actions from Etherscan still goes against Etherscan's terms [2], and it potentially raises questions about the ethicality of the collection process for DeFi actions from Etherscan.

8 RELATED WORK

We introduced four categories of closely-related work.

DeFi action identification. Majority of existing studies [18, 70–72, 82, 84, 85] only focus on a few DeFi applications and could not cover other DeFi applications. We compared them in §1 and §3.2. Etherscan [2] identifies 7 kinds of DeFi actions, and DeFiRanger [89] automatically recognizes DeFi actions. However, both of them suffer inaccurate results, and our approach outperforms them (§5.3).

Design on extracting MEV. Eskandari et al. [44] introduce the front-running taxonomy. Zhou et al. [96] generate profitable MEV activities by interacting with AMMs. Zhou et al. [98] formalize Sandwich Attacks with crafted Swap actions on AMMs. Several studies (e.g., [33, 73]) model specific kinds of MEV activities, and determine optimal parameters to maximize the revenue of extracting MEV. None of them can be used to conduct a systematic study on DeFi MEV activities, because they cannot recognize DeFi MEV activities with unknown patterns of DeFi actions.

MEV evaluation. Existing studies only quantify known MEV activities, and cannot discover unknown MEV activities. Torres et al. [47] measure three types of front-running. Daian et al. [40] evaluate the front-running under the gas price auction. Qin et al. [72] quantify five kinds of MEV activities. For known MEV activities, several studies [62, 70, 71, 82, 83, 85] evaluate their impact, users' perceptions of them, and their prevalence in private transactions.

MEV mitigation. Researchers propose countermeasures to mitigate threats caused by known MEV activities, and our insights from new DeFi MEV activities can contribute to them. One solution is to guarantee the transaction order fairness (e.g., [54, 55]) so that validators/miners and traders cannot modify transaction positions to extract MEV. Other studies propose new blockchain platforms or applications to prevent front-running [36, 52, 59, 97]. Furthermore, several studies [34, 53, 93] systematize countermeasures against the front-running, transaction reordering manipulation, and MEV, and discuss the corresponding attacks and open challenges.

9 CONCLUSION

We conduct the first systematic study on DeFi MEV activities in Flashbots bundle by developing ACTLIFTER, a novel automated tool for accurately identifying DeFi actions in transactions, and ACTCLUSTER, a new approach that leverages iterative clustering to facilitate the discovery of DeFi MEV activities. Our experimental results show that ACTLIFTER achieves nearly 100% accuracy in identifying DeFi actions, significantly outperforming existing techniques. With the help of ACTCLUSTER, we discover 17 new kinds of DeFi MEV activities, which occur in 53.12% of bundles but have not been reported. Moreover, we demonstrate that ACTLIFTER and ACTCLUSTER are very useful in MEV studies by six applications.

REFERENCES

[1] 2013. EVM RPC APIs. https://geth.ethereum.org/docs/rpc/ns-debug.
[2] 2015. Etherscan. https://etherscan.io.
[3] 2018. Ampleforth Token. https://www.ampleforth.org/.
[4] 2018. chiru-labs ERC721. https://github.com/chiru-labs/ERC721A.
[5] 2018. ERC721. https://github.com/OpenZeppelin/openzeppelin-contracts.
[6] 2019. Compound. https://compound.finance/.
[7] 2019. Ethereum Dapps Ranking. https://www.dapp.com.
[8] 2019. The Maker Foundation. https://makerdao.com/en/.
[9] 2020. Aave Protocol. https://homora-v2.alphafinance.io/.
[10] 2020. DefiLlama. https://defillama.com/.
[11] 2021. Bloxroute relay. https://docs.bloxroute.com/apis/mev-solution.
[12] 2021. DeFiPulse. https://www.defipulse.com/.
[13] 2021. Eden relay. https://relay.edennetwork.io/info.
[14] 2021. EthTx. https://github.com/EthTx/ethtx.
[15] 2021. Flashbots API. https://blocks.flashbots.net/.
[16] 2021. Flashbots Bundle. https://docs.flashbots.net.
[17] 2021. The Graph. https://thegraph.com/explorer/.
[18] 2021. MEV-Explore. https://explore.flashbots.net/.
[19] 2021. mev-inspect. https://github.com/flashbots/mev-inspect-py.
[20] 2021. Multi-layered Burger Arbitrage. twitter.com/bertcmiller/....
[21] 2021. One FN transaction of Etherscan. 0x9a7c52f6...e57d9400.
[22] 2022. Ethereum Merge. https://ethereum.org/en/upgrades/merge/.
[23] 2022. flashbots mevboost dashboard. https://boost-relay.flashbots.net/.
[24] 2022. MEV relay list. https://github.com/eth-educators/ethstaker-guides.
[25] 2022. mevboost. https://boost.flashbots.net/.
[26] 2022. mevboost dashboard. https://dune.com/ChainsightAnalytics.
[27] 2022. Solidity document. https://docs.soliditylang.org/en/latest.
[28] 2023. MEVBlocker. https://mevblocker.io/.
[29] 2023. Our full paper with the appendix. https://zzzihao-li.github.io/.
[30] Hayden Adams, Noah Zinsmeister, and Dan Robinson. 2020. Uniswap v2 Core.
[31] Kushal Babel, Philip Daian, Mahimna Kelkar, and Ari Juels. 2023. Clockwork finance: Automated analysis of economic security in smart contracts. In *SP*.
[32] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch Lafuente. 2021. SoK: lending pools in decentralized finance. In *FC*.
[33] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. 2022. Maximizing Extractable Value from Automated Market Makers. *FC* (2022).
[34] Carsten Baum, James Hsin-yu Chiang, Bernardo David, Tore Kasper Frederiksen, and Lorenzo Gentile. 2021. SoK: Mitigation of Front-running in Decentralized Finance. *ePrint* (2021).
[35] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE TPAMI* (2013).

[36] Lorenz Breidenbach, Phil Daian, Florian Tramèr, and Ari Juels. 2018. Enter the Hydra: Towards Principled Bug Bounties and {Exploit-Resistant} Smart Contracts. In *USENIX Security*.

[37] Ting Chen, Zihao Li, Xiapu Luo, Xiaofeng Wang, Ting Wang, Zheyuan He, Kezhao Fang, Yufei Zhang, Hang Zhu, Hongwei Li, et al. 2021. SigRec: Automatic Recovery of Function Signatures in Smart Contracts. *IEEE TSE* (2021).

[38] Ting Chen, Zihao Li, Yufei Zhang, Xiapu Luo, Ang Chen, Kun Yang, Bin Hu, Tong Zhu, Shifang Deng, Teng Hu, et al. 2019. Dataether: Data exploration framework for ethereum. In *IEEE ICDCS*.

[39] Ting Chen, Yufei Zhang, Zihao Li, Xiapu Luo, Ting Wang, Rong Cao, Xiuzhuo Xiao, and Xiaosong Zhang. 2019. Tokenscope: Automatically detecting inconsistent behaviors of cryptocurrency tokens in ethereum. In *ACM CCS*.

[40] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2020. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *IEEE SP*.

[41] Francesco D'Amato, Joachim Neu, Ertem Nusret Tas, and David Tse. 2022. No More Attacks on Proof-of-Stake Ethereum? *arXiv* (2022).

[42] Chris Dannen. 2017. *Introducing Ethereum and solidity*.

[43] Dipanjan Das, Priyanka Bose, Nicola Ruaro, Christopher Kruegel, and Giovanni Vigna. 2021. Understanding Security Issues in the NFT Ecosystem. *arXiv* (2021).

[44] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. 2019. Sok: Transparent dishonesty: front-running attacks on blockchain. In *FC*.

[45] Vogelsteller Fabian and Buterin Vitalik. 2015. EIP-20: Token Standard. https://eips.ethereum.org/EIPS/eip-20.

[46] Luca Faes, Giandomenico Nollo, Sebastiano Stramaglia, and Daniele Marinazzo. 2017. Multiscale granger causality. *Physical Review E* (2017).

[47] Christof Ferreira Torres, Ramiro Camino, and Radu State. 2021. Frontrunner Jones and the Raiders of the Dark Forest: An Empirical Study of Frontrunning on the Ethereum Blockchain. In *USENIX Security*.

[48] Michael Fröwis, Andreas Fuchs, and Rainer Böhme. 2019. Detecting token systems on ethereum. In *FC*.

[49] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM CSUR* (2014).

[50] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. 2018. Recent advances in convolutional neural networks. *Pattern recognition* (2018).

[51] Zheyuan He, Shuwei Song, Yang Bai, Xiapu Luo, Ting Chen, Wensheng Zhang, Peng He, Hongwei Li, Xiaodong Lin, and Xiaosong Zhang. 2023. TokenAware: Accurate and efficient bookkeeping recognition for token smart contracts. *TOSEM* (2023).

[52] Lioba Heimbach and Roger Wattenhofer. 2022. Eliminating Sandwich Attacks with the Help of Game Theory. *arXiv* (2022).

[53] Lioba Heimbach and Roger Wattenhofer. 2022. SoK: Preventing Transaction Reordering Manipulations in Decentralized Finance. *arXiv* (2022).

[54] Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. 2021. Themis: Fast, Strong Order-Fairness in Byzantine Consensus. *ePrint* (2021).

[55] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. 2020. Order-fairness for byzantine consensus. In *CRYPTO*.

[56] Selim Kılıç. 2015. Kappa testi. *Journal of Mood Disorders* (2015).

[57] Klaus Krippendorff. 2011. Computing Krippendorff's alpha-reliability. (2011).

[58] Shipeng Li, Jingwei Li, Yuxing Tang, Xiapu Luo, Zheyuan He, Zihao Li, Xi Cheng, Yang Bai, Ting Chen, Yuzhe Tang, et al. 2023. BlockExplorer: Exploring Blockchain Big Data via Parallel Processing. *TC* (2023).

[59] LibSubmarine. 2019. Defeat Front-Running. https://libsubmarine.org/.

[60] Jinyang Liu, Jieming Zhu, Shilin He, Pinjia He, Zibin Zheng, and Michael R Lyu. 2019. Logzip: extracting hidden structures via iterative clustering for log compression. In *Proc. ASE*.

[61] Yulin Liu, Yuxuan Lu, Kartik Nayak, Fan Zhang, Luyao Zhang, and Yinhong Zhao. 2022. Empirical analysis of eip-1559: Transaction fees, waiting time, and consensus security. In *CCS*.

[62] Xingyu Lyu, Mengya Zhang, Xiaokuan Zhang, Jianyu Niu, Yinqian Zhang, and Zhiqiang Lin. 2022. An Empirical Study on Ethereum Private Transactions and the Security Implications. *arXiv* (2022).

[63] Daniel Macrinici, Cristian Cartofeanu, and Shang Gao. 2018. Smart contract applications within blockchain technology: A systematic mapping study. *Telematics and Informatics* (2018).

[64] Leann Myers and Maria J Sirois. 2004. Spearman correlation coefficients, differences between. *Encyclopedia of statistical sciences* (2004).

[65] Joachim Neu, Ertem Nusret Tas, and David Tse. 2022. Two More Attacks on Proof-of-Stake GHOST/Ethereum. In *ConsensusDay*.

[66] Michael Neuder, Daniel J Moroz, Rithvik Rao, and David C Parkes. 2021. Low-cost attacks on Ethereum 2.0 by sub-1/3 stakeholders. *WINE* (2021).

[67] Mehdi Noroozi, Hamed Pirsiavash, and Paolo Favaro. 2017. Representation learning by learning to count. In *IEEE ICCV*.

[68] David L Olson and Dursun Delen. 2008. *Advanced data mining techniques*.

[69] Michael Pacheco, Gustavo A Oliva, Gopi Krishnan Rajbahadur, and Ahmed E Hassan. 2022. Is my transaction done yet? An empirical study of transaction processing times in the Ethereum Blockchain Platform. *TOSEM* (2022).

[70] Julien Piet, Jaiden Fairoze, and Nicholas Weaver. 2022. Extracting Godl [sic] from the Salt Mines: Ethereum Miners Extracting Value. *arXiv* (2022).

[71] Kaihua Qin, Liyi Zhou, Pablo Gamito, Philipp Jovanovic, and Arthur Gervais. 2021. An empirical study of defi liquidations: Incentives, risks, and instabilities. In *ACM IMC*.

[72] Kaihua Qin, Liyi Zhou, and Arthur Gervais. 2022. Quantifying Blockchain Extractable Value: How dark is the forest? *IEEE SP* (2022).

[73] Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. 2021. Attacking the defi ecosystem with flash loans for fun and profit. In *FC*.

[74] Dennis W Ruck, Steven K Rogers, and Matthew Kabrisky. 1990. Feature selection using a multilayer perceptron. *Journal of Neural Network Computing* (1990).

[75] Fabian Schär. 2021. Decentralized finance: On blockchain-and smart contract-based financial markets. *FRB of St. Louis Review* (2021).

[76] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. 2019. Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In *ACM CCS*.

[77] Sergei Tikhomirov. 2017. Ethereum: state of knowledge and research perspectives. In *FPS*.

[78] Hiro Y Toda and Taku Yamamoto. 1995. Statistical inference in vector autoregressions with possibly integrated processes. *Journal of econometrics* (1995).

[79] Talip Ucar, Ehsan Hajiramezanali, and Lindsay Edwards. 2021. Subtab: Subsetting features of tabular data for self-supervised representation learning. In *NeurIPS*.

[80] Friedhelm Victor. 2020. Address clustering heuristics for Ethereum. In *FC*.

[81] Weiran Wang, Raman Arora, Karen Livescu, and Jeff Bilmes. 2015. On deep multi-view representation learning. In *ICML*.

[82] Ye Wang, Yan Chen, Shuiguang Deng, and Roger Wattenhofer. 2021. Cyclic Arbitrage in Decentralized Exchange Markets. *WWW* (2021).

[83] Ye Wang, Patrick Zuest, Yaxing Yao, Zhicong Lu, and Roger Wattenhofer. 2022. Impact and User Perception of Sandwich Attacks in the DeFi Ecosystem. In *CHI*.

[84] Zhipeng Wang, Kaihua Qin, Duc Vu Minh, and Arthur Gervais. 2022. Speculative Multipliers on DeFi: Quantifying On-Chain Leverage Risks. *FC* (2022).

[85] Ben Weintraub, Christof Ferreira Torres, Cristina Nita-Rotaru, and Radu State. 2022. A Flash (bot) in the Pan: Measuring Maximal Extractable Value in Private Pools. In *IMC*.

[86] Sam M Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J Knottenbelt. 2021. Sok: Decentralized finance. In *arXiv*.

[87] Entriken William, Shirley Dieter, Evans Jacob, and Sachs Nastassia. 2018. EIP-721 Non-Fungible Token Standard. https://eips.ethereum.org/EIPS/eip-721.

[88] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* (2014).

[89] Siwei Wu, Dabao Wang, Jianting He, Yajin Zhou, Lei Wu, Xingliang Yuan, Qinming He, and Kui Ren. 2021. DeFiRanger: Detecting Price Manipulation Attacks on DeFi Applications. *arXiv* (2021).

[90] Zhiwen Xie, Runjie Zhu, Kunsong Zhao, Jin Liu, Guangyou Zhou, and Jimmy Xiangji Huang. 2021. Dual gated graph attention networks with dynamic iterative training for cross-lingual entity alignment. *TOIS* (2021).

[91] Jiahua Xu, Nazariy Vavryk, Krzysztof Paruch, and Simon Cousaert. 2023. SoK: Decentralized Exchanges (DEX) with Automated Market Maker (AMM) protocols. *Comput. Surveys* (2023).

[92] Rui Xu and Donald Wunsch. 2005. Survey of clustering algorithms. *TNN* (2005).

[93] Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. 2022. SoK: MEV Countermeasures: Theory and Practice. *arXiv* (2022).

[94] Wuqi Zhang, Lili Wei, Shuqing Li, Yepang Liu, and Shing-Chi Cheung. 2021. Darcher: Detecting on-chain-off-chain synchronization bugs in decentralized applications. In *ESEC/FSE*.

[95] Kunsong Zhao, Zihao Li, Jianfeng Li, He Ye, Xiapu Luo, and Ting Chen. 2023. DeepInfer: Deep Type Inference from Smart Contract Bytecode. In *ESEC/FSE*.

[96] Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, and Arthur Gervais. 2021. On the just-in-time discovery of profit-generating transactions in defi protocols. In *IEEE SP*.

[97] Liyi Zhou, Kaihua Qin, and Arthur Gervais. 2021. A2mm: Mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges. In *arXiv*.

[98] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. 2021. High-frequency trading on decentralized on-chain exchanges. In *IEEE SP*.

[99] Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. 2022. SoK: Decentralized Finance (DeFi) Incidents. *arXiv* (2022).

[100] Hao Zhu, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. 2017. Iterative entity alignment via knowledge embeddings. In *IJCAI*.

A EXAMPLE OF EVENT EXTRACTION

Fig. 10a shows Uniswap's [30] descriptions of `Swap` event. We confirm it corresponds to a Swap action by its descriptions in Line 8. Fig. 10b shows code snippets and comments of another `Swap` event and swap function from Smoothy (https://smoothy.finance/). We confirm it corresponds to a Swap action by comments in Line 2 and the two functions which will trigger asset transfers in Line 5 and 6.

```
1  event Swap {
2      address indexed sender,
3      uint amount0In,
4      uint amount1In,
5      uint amount0Out,
6      uint amount1Out,
7      address indexed to};
8  // Emitted each time a swap occurs via swap function.
```

(a) descriptions of `Swap` event in Uniswap's document

```
1  event Swap{...};
2  //* @dev Swap a token to another.
3  function swap(...){
4      ...
5      _transferIn(infoIn, bTokenInAmount);
6      _transferOut(infoOut, bTokenOutAmount, adminFee);
7      emit Swap(...);}
```

(b) code snippets and comments of `Swap` event in Smoothy's codes

Figure 10: Event information of Uniswap and Smoothy.

B RECOGNIZE ASSET TRANSFERS

• **Token transfer.** In a token transfer $Asset_C.Transfer(From, To, Value)$, $From$ sends $Value$ amounts of $Asset_C$ to To. Hence, c_1, denoted as $C.Event(Transfer(From,To,Value))$, checks whether a token transfer occurs when a `Transfer` event is emitted by C with the parameters of $From$, To, and $Value$. A token transfer should satisfy three requirements in c_2: i) $From$ is not the zero address and C's address (i.e., $From \notin (0x00...00, C)$), and To is also not the zero address and C's address (i.e., $To \notin (0x00...00, C)$). This requirement is based on the widely used templates for ERC20 and ERC721 (e.g., OpenZeppelin [5] and chiru-labs [4]). In their templates, the zero address and the address of C are used for token minting and burning. ii) the amount of transferred token $Value$ is non-zero (i.e., $Value \neq 0$), and iii) $From$ and To are different addresses (i.e., $From \neq To$). Note that there is no actual asset transfer between $From$ and To if any of the last two requirements are violated.

• **ERC721 token minting/burning.** In an ERC721 token minting (resp. burning) $Asset_C^{721}.Minting(From, To, Value)$ (resp. $Asset_C^{721}.Burning(From, To, Value)$), the ERC721 token contract C mints (resp. burns) an NFT with the tokenId $Value$. Hence, c_1, denoted as $C.Event(Transfer(From,To,Value))$, checks whether an ERC721 token minting (resp. burning) occurs when C emits a `Transfer` event with the parameters of $From$, To, and $Value$. An ERC721 token minting (resp. burning) should satisfy two requirements in c_2: i) $From$ (resp. To) is the zero address or C's address (i.e., $From$ (resp. To) $\in (0x00...00, C)$), and To (resp. $From$) is not the zero address and C's address (i.e., To (resp. $From$) $\notin (0x00...00, C)$). This requirement is based on the widely used templates for ERC721 (e.g., OpenZeppelin [5] and chiru-labs [4]). In their templates, the zero address and C's address are used for ERC721 token minting and burning. ii) C implements standard functions defined in ERC721 (i.e., $C \models ERC721\ standard$), and thus, $Asset_C$ is an ERC721 asset.

C ALTERNATIVE APPROACH OF ACTLIFTER.

To evaluate the effectiveness of collected events in $\mathbb{M}$, we created two variants of ACTLIFTER. ACTLIFTER$_{a1}$ replaces $\mathbb{M}$ (§3.2) with other information collected more automatically, i.e., contract addresses of DeFi applications. Besides, in **S-1**, ACTLIFTER$_{a1}$ recognizes asset transfers involved in DeFi actions, if the contract addresses of DeFi applications receive or send assets in the asset transfers. Then ACTLIFTER$_{a1}$ identifies DeFi actions in **S-2**. To obtain the contract addresses, we queried the APIs of graph, which provides blockchain data to developers [17]. ACTLIFTER$_{a2}$ ignores $\mathbb{M}$, only recognizes all asset transfers in transactions in **S-1**, and identifies DeFi actions in **S-2**.

We compare ACTLIFTER with its two variants by 500 transactions. It shows that ACTLIFTER$_{a1}$ reports 7 false actions, because asset transfers can simultaneously satisfy asset transfer patterns of different kinds of actions, and a DeFi contract can perform different kinds of actions. By only using address information, ACTLIFTER$_{a1}$ cannot distinguish DeFi actions performed by the contract. ACTLIFTER$_{a2}$ reports 87 false actions performed by non-DeFi contracts due to wrongly pairing asset transfers, because ACTLIFTER$_{a2}$ identifies DeFi actions only according to asset transfer patterns. By contrast, ACTLIFTER correctly identifies all DeFi actions with $\mathbb{M}$.

D FREQUENCY OF EVENTS IN $\mathbb{M}$

To ensure comprehensive and reliable results, we further assess whether all 88 events in $\mathbb{M}$ (§3.2) occur in D_{Bundle}. Fig. 11 displays the frequency of each event in $\mathbb{M}$ (§3.2) that occurs in D_{Bundle}. We find that all 88 events are covered, with 64 of them (72.7%) occurring at least 100 times in D_{Bundle}. Additionally, the results indicate that methods (e.g., [18, 70–72, 82, 84, 85] in Table 1) relying on several specific events to identify DeFi actions will miss reporting a significant number of DeFi actions.

E NEW MEV ACTIVITIES IN BUNDLES

We discover 17 kinds of new DeFi MEV activities in bundles, which are summarized in Table 6. For a more thorough understanding of them, we refer readers to the previous helpful studies [47, 71, 82] which provide basic descriptions for the three known MEV activities, i.e., Sandwich Attack, Cyclic Arbitrage, and Liquidation.

E.1 Multi-layered Burger Arbitrage (MBA)

It involves more than three transactions in a bundle. The first and last transactions are emitted by the arbitrageur as A_1 and A_2, and all the other transactions are emitted by other traders as $V_1,...,V_n$, where $n > 1$. All V_i and A_1 aim to trade X asset for Y asset in the same AMM, and A_2 aims to trade Y for X with the same AMM. The Multi-layered Burger Arbitrage is similar to Sandwich Attack [72],
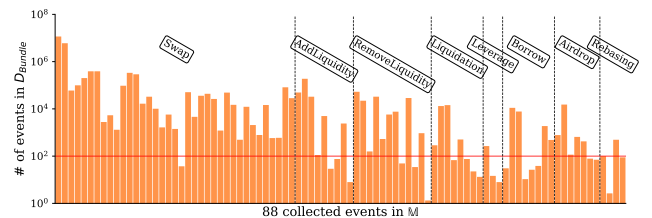

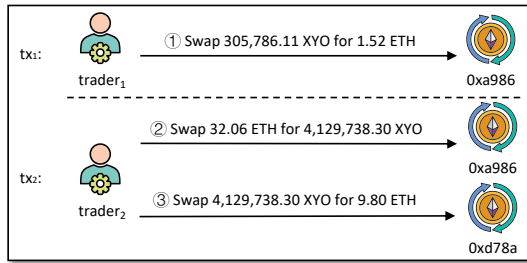**Figure 11: Frequency of events in $\mathbb{M}$**

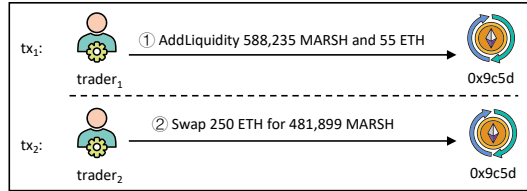**Figure 13: An example of Failed Arbitrage**

Figure 14: An example of Liquidity Backrun Arbitrage

except that there are more than one transaction in the middle of A_1 and A_2 transactions. All V_i are used to pull up the price of Y in the AMM, and further improve the arbitrageur's revenue. For example, an arbitrageur performed the Multi-layered Burger Arbitrage in the first bundle of the 12,753,463 block [20]. In the bundle, there are 4 victim transactions. The four victim transactions all trade ETH for F9 with AMM 0x459e. The arbitrageur earns 1.16 ETH as profits by the A_1 and A_2 in the bundle.

E.2 Liquidity Sandwich Arbitrage (LSA)

It involves three transactions. The first and third transactions are signed by the trader$_1$ as A_1 and A_2, and the second transaction is signed by trader$_2$ as V. The V aims to trade X asset for Y asset in an AMM, A_1 and A_2 aim to supply and withdraw X and Y assets with the same AMM. Different from Sandwich Attack [72], Liquidity Sandwich Arbitrage does not aim to pull up the price of the traded assets like Y when V executes, but the trader$_1$ aims to be the liquidity provider to earn the exchange fee for the swapping in V. In fact, V can set the slippage protection parameter [52] to require the minimum amount of received assets, if trader$_1$ conducts the Sandwich Attack, the slippage protection parameter can trigger and the execution of V will revert. Hence the Sandwich Attack gains no profits. For example, Fig. 12 shows the three transactions in the first bundle of the 12,702,238 block. In the first and third transactions, the trader$_1$ supplies and withdraws USDC and ETH with the AMM 0x8ad5, respectively, and in the second transaction, trader$_2$ trades USDC for ETH in the AMM 0x8ad5. According to the price in Etherscan [2] of ETH and USDC on the day of mining the 12,702,238 block, trader$_1$ earns profits of 18637.5 USD.

Insight. The observation from LSA yields the security insights for MEV countermeasures implemented in the contracts (e.g., slippage protection [52], atomic routing [97], and optimal slippage setting [52]). More precisely, these MEV countermeasures rely on the parameters in contracts to defend against MEV (e.g., failing the transactions where parameters are triggered). However, bundle arbitrageurs can still maximize their revenue without triggering

the MEV protection mechanisms implemented in the contracts, because bundle arbitrageurs can manipulate the order of transactions in their bundles to make their arbitrage transactions and victim transactions execute in bundle arbitrageurs' expected order.

E.3 Backrun Cyclic Arbitrage (BCA)

It involves two transactions. In the first one, trader$_1$ performs Swap, AddLiquidity, or RemoveLiquidity actions, the actions trigger the unbalanced prices among the AMMs [72]. In the second one, trader$_2$
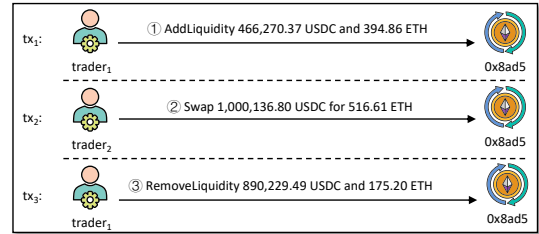

Figure 12: An example of Liquidity Sandwich Arbitrage

backruns the former transaction to gain profit among the AMMs by Cyclic Arbitrage.

E.4 Failed Arbitrage (FA)

For the Failed Arbitrage, the arbitrageur aims to obtain profits by performing Sandwich Attack or Cyclic Arbitrage in a bundle, and suffers the financial loss. For example, Fig. 13 shows the two transactions in the second bundle of the 12,516,458 block. In the first transaction, trader$_1$ trades XYO for ETH in AMM 0xa986. In the second transaction, trader$_2$ aims to conduct the Cyclic Arbitrage to backrun the first transaction in the AMM 0xa986. Unfortunately, the trader$_2$ suffers the financial loss of 22.26 (32.06-9.8) Ether due to trading XYO for ETH in the AMM 0xd78a.

E.5 Hybrid Arbitrage (HA)

For the Hybrid Arbitrage, there are at least two kinds of MEV activities of the three known MEV activities (i.e., Sandwich Attack, Cyclic Arbitrage, and Liquidation) in a bundle. Besides, the occurrence of transactions of the MEV activities is crossed. Hybrid Arbitrage activities are the cases when arbitrageurs perform multiple kinds of MEV activities in a bundle. Besides, to minimize the transaction fee cost, arbitrageurs merge multiple kinds of MEV activities into a single transaction. For example, in a transaction, an arbitrageur first performs a Liquidation activity to receive the collateral assets, and then uses received assets to perform a Cyclic Arbitrage activity.

E.6 Swap Backrun Arbitrage (SBA)

It involves two transactions. The former executes a Swap action to exchange X asset for Y asset in an AMM which pulls up Y's price, and the latter backruns the former transaction by exchanging Y asset for X asset in the same AMM to sell Y at the higher price than without the former transaction.

E.7 Liquidity Backrun Arbitrage (LBA)

It involves two transactions. The former executes an AddLiquidity/RemoveLiquidity action on an AMM which causes the unbalanced prices between AMMs, and the latter executes a Swap action
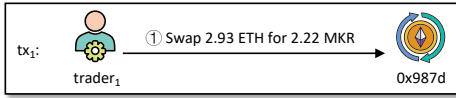
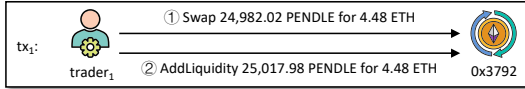**Figure 15: An example of Non-cyclic Swap Trade**

Figure 16: An example of Liquidity-swap Trade

to trade the corresponding assets on the same AMM to obtain profits from the price differences. For example, Fig. 14 shows the two transactions in the first bundle of the 12,141,301 block. In the former transaction, $trader_1$ supplies MARSH and ETH to the AMM 0x9c5d. In the latter transaction, $trader_2$ performs a Swap to trade ETH for MARSH with the same AMM. According to the price in Etherscan [2] of MARSH and ETH on the day of mining the 12,141,301 block, in the latter transaction, the $trader_2$ trades ETH for MARSH which are worth 738,060 and 3,504,910.65 USD, respectively.

E.8 Liquidity-swap Trade (LT)

For the Liquidity-swap Trade, there exists a transaction that a trader both trades assets on AMMs and performs the AddLiquidity or RemoveLiquidity actions on AMMs. There are two cases for the Liquidity-swap Trade, i) the trader trades assets and supplies the traded assets into an AMM at the expected price, ii) the trader withdraws assets from an AMM and trades the returned assets. For example, Fig. 16 shows the transaction in the second bundle of the 13,521,679 block, the trader trades PENDLE for ETH and supplies PENDLE and the traded ETH to the AMM 0x3792. By the transaction, the trader becomes the liquidity provider at the expected price of assets in the AMM 0x3792.

E.9 Partial Cyclic Arbitrage (PCA)

For the Partial Cyclic Arbitrage, there exists a transaction that performs multiple Swap actions among AMMs. Part of the Swap actions can fit into a single cycle one by one in the transaction. The Partial Cyclic Arbitrage distinguishes the Cyclic Arbitrage, because Cyclic Arbitrage only considers the transactions in which all the Swap actions fit into a single cycle.

E.10 Non-cyclic Swap Trade (NST)

For the Non-cyclic Swap Trade, the transactions in bundles only perform the Swap actions among AMMs. Besides, there are no known MEV activities, i.e., Sandwich Attack, Cyclic Arbitrage, and Liquidation, no the other 9 kinds of DeFi actions, and no other 16 kinds of new MEV activities. The trader who performs the Non-cyclic Swap Trade in a bundle aims to trade on the AMMs at the expected price. For example, Fig. 15 shows the transaction in the first bundle of the 12,244,578 block. The $trader_1$ only trades ETH for MKR with the AMM 0x987d. According to the price in Etherscan [2] of ETH and MKR on the day of mining the 12,244,578 block, the $trader_1$ trades ETH for MKR which are worth 7,356.07 and 8,375.68 USD, respectively. Hence, $trader_1$ obtains profits as 1,019.61 USD.

E.11 Bulk NFT-Minting (BN)

It only involves transactions that mint NFTs. There are two cases for the Bulk NFT-Minting, i) the NFT contract mints multiple NFTs in a single transaction, ii) the NFT contract mints multiple NFTs among multiple transactions. For example, there is one transaction in the second bundle of the 13,336,591 block, and the VIXEN NFT contract mints 20 NFTs.

E.12 NFT Reforging (NR)

It involves one transaction, where the NFT contract first burns one NFT and then remints the NFT with the same tokenId and the specific asset represented by the NFT is updated. For example, the tenth bundle at 14,579,991 block contains one transaction, where the resolved address of the ENS NFT with the name blockchainpolice.eth is updated to address 0x6669.

E.13 Airdrop Claiming (AC)

For the Airdrop Claiming, a trader only claims airdrop rewards in transactions of a bundle.

E.14 NFT-Minting-swap Trade (NT)

It involves a transaction where a trader first receives a minted NFT, then the trader conducts asset exchange to sell the minted NFT to an AMM.

E.15 Loan-powered Arbitrage (LA)

It involves a transaction where an arbitrageur first loans assets from Lending under the over/under-collateral deposit, then uses the loaned assets to conduct MEV activities, e.g., Cyclic Arbitrage.

E.16 Airdrop-swap Trade (AT)

It involves a transaction, where a trader both claims the airdrop rewards and sells the received assets to an AMM by swapping.

F BUNDLE FORMATTING

• **Action block.** An action block consists of parameters to characterize the corresponding type of action. It is a matrix where a column vector corresponds to either a parameter or a separator. Specifically, the first column vector of each action block is the action header and it represents the action type with one-hot encoding. The second column vector acts as a separator between the action header and other parameters, and all entries in it equal to -1. We index all addresses involved in a transaction in chronological order and represent them using their indices instead of the original addresses. We also index different assets according to their popularity. It is worth noting that the transferred amounts of assets are normalized in the range $[-1, 1]$ to avoid huge differences in parameter scale.

• **Transaction block.** It is comprised of meta information of a transaction and all actions in it. The first (resp. second) column vector of a transaction block represents the sender (resp. recipient) of this transaction. The last column vector records asset changes for different addresses after the transaction. Between the second and last column vectors, action blocks are sequentially concatenated and connected by separators.
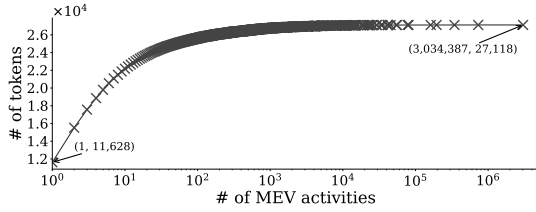
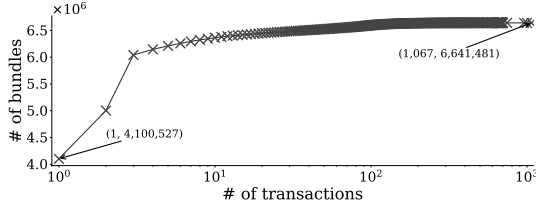**Figure 18: Number of tokens involved in new MEV activities.**

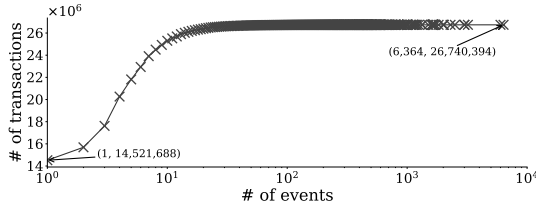**Figure 19: The number of transactions in each bundle.**

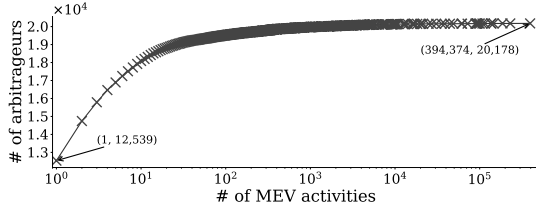**Figure 20: Number of events in each transaction of bundles.**

Figure 17: Number of contracts invoked directly by arbitrageurs to perform new DeFi MEV activities.

• **Bundle matrix.** It is constructed by combining all transaction blocks within a bundle. Transaction blocks are arranged in chronological order to reflect the temporal patterns related to DeFi actions in a bundle. Successive transaction blocks are separated by two separators. The height of bundle matrices equals the length of the column vector, whereas the width of bundle matrices can be variable depending on the number of encapsulated transaction blocks. To facilitate feature extraction and feature representation, we fix the width of bundle matrices by specifying the maximum width. The bundle matrix will be truncated if its width exceeds the maximum width, otherwise, it will be padded with -1 entries to fit the maximum width.

G PREVALENCE OF NEW MEV ACTIVITIES

Table 8 shows the number of bundles for each kind of MEV activities. The first row lists the type of DeFi MEV activities, and the second row lists the number of bundles containing corresponding DeFi MEV activities.

• The number of bundles that contain different counts of new MEV types. 3,459,988 bundles only contain one kind of new MEV

activities while 63,851 (resp. 3,816) bundles contain two (resp. three) kinds of new MEV activities.

• The number of contracts that are directly invoked by the EOA account of arbitrageurs to perform new DeFi MEV activities. In Fig. 17, each cross (x, y) indicates that y contracts are invoked by arbitrageurs to conduct no more than x MEV activities in all bundles. It shows that 62.14% (12,539/20,178) of contracts are involved in new DeFi MEV activities only once. Table 9 lists the top 5 contracts that are involved in new DeFi MEV activities. The first, second, and third rows list the contract addresses, the number of involved MEV activities, and the labels in Etherscan, respectively. The first contract is labeled as MEV Bot by Etherscan, and the first contract is directly invoked by arbitrageurs to conduct 11.18% (394,374/3,527,655) of bundles containing new DeFi MEV activities. The other four contracts are labeled as AMM routers by Etherscan. It shows that the bundle arbitrageurs can calculate the parameters to perform MEV activities offline, and directly invoke the AMM routers to perform MEV activities.

• The number of tokens used to perform new DeFi MEV activities by arbitrageurs. In Fig. 18, each cross (x, y) indicates that y tokens are involved in no more than x MEV activities in all bundles. It shows that 42.88% (11,628/27,118) of tokens are involved in new DeFi MEV activities only once. Table 7 lists the top 10 tokens involved in

Table 7: Top 10 tokens involved in new MEV activities

Token	WETH	USDC	USDT	WBTC	DAI	BNT	SHIB	LINK	APE	SUSHI
# MEV activities	3,034,387	730,832	344,353	195,240	162,178	80,408	55,607	54,569	45,371	42,014

new DeFi MEV activities. The first and second rows list the tokens, and the number of involved MEV activities. It shows that WETH is involved in most of the new DeFi MEV activities (i.e., 86.02%).

• The number of transactions in each bundle. In Fig. 19, each cross (x, y) indicates that y bundles contain no more than x transactions. 61.74% bundles have only one transaction. Besides, the fourth bundle in the 13,143,462 block mined by F2Pool contains the most amount of transactions (i.e., 1,067). In the bundle, F2Pool distributes ETH to different EOA accounts for distributing mining revenues[1].

• The number of events in each transaction of bundles. In Fig. 20, each cross (x, y) indicates that y transactions contain no more than x events. For 54.31% transactions there is no more than one event in it, and for 31.81% (8,506,831/26,740,394) transactions there is no event. We find that the transaction[2] emits the most amount of events (i.e., 6,364). In the transaction, an NFT contract mints 6,363 NFTs. For each minted NFT, the contract emits a `Transfer` event.

FP/FN rates. We further evaluate the FP/FN rates for the results of new DeFi MEV activity in Table 8. Due to the lack of ground truth for MEV activities, we can only evaluate the FP/FN by manually checking each bundle. Since to manually analyze all bundles are labor-intensive, we choose to sample bundles from D_{Bundle} and manually evaluate the FP/FN results of the sampled bundles. Specifically, for each kind of MEV activity, we sample 20 bundles from bundles which contain such kind of MEV activity, and we sample 20 bundles from bundles that do not contain the MEV activity. The results show that there is no FP or FN for results of new DeFi MEV activities in our sampled bundles.

---

[1] https://f2pool.io/mining/guides/how-to-mine-ethereumpow/
[2] 0xa90088e0...af1b5911

Table 8: Number of bundles for each kind of DeFi MEV activities

MEV type	SA	CA	LI	SBA	LBA	LSA	MBA	LT	PCA	BCA	HA	FA	NST	RBA	AT	BN	NR	AC	NT	LA
# bundles	813,188	1,334,207	14,263	162,375	5,045	12,830	3,654	5,578	65,670	46,771	70,413	46,784	3,160,094	54	128	16,327	218	2,388	562	2,470

SA: Sandwich Attack, CA: Cyclic Arbitrage, LI: Liquidation, SBA: Swap Backrun Arbitrage, LBA: Liquidity Backrun Arbitrage, LSA: Liquidity Sandwich Arbitrage, MBA: Multi-layered Burger Arbitrage, LT: Liquidity-swap Trade, PCA: Partial Cyclic Arbitrage, BCA: Backrun Cyclic Arbitrage, HA: Hybrid Arbitrage, FA: Failed Arbitrage, NST: Non-cyclic Swap Trade, RBA: Rebasing Backrun Arbitrage, AT: Airdrop-swap Trade, BN: Bulk NFT-Minting, NR: NFT Reforging, AC: Airdrop Claiming, NT: NFT-Minting-swap Trade, LA: Loan-powered Arbitrage.

Table 10: Three DeFi incidents found during the procedure of discovering DeFi MEV activities

Transaction	Financial losses (USD)	Description
0xa9a1b8ea...d7ffe61e	18.8M	Cream Finance exploitation
0x7cc7d935...9840da22	24.5M	xToken exploitation
0x5a6c108d...dc3bfc1f	42.2K	RigoBlock whitehat rescue

```
1   /// @dev Allows owner to set allowances to multiple approved
        tokens with one call.
2   function setMultipleAllowances(...) {...}
3   ...
4   /// @dev Allows owner to operate on exchange through extension.
5   function operateOnExchange(...) onlyOwner {...}
```

Figure 21: Code snippets of RigoBlock contract.

Table 9: Top 5 contracts involved in new MEV activities

Contract	0xa57b...d6cf	0x7a25...488d	0x68b3...fc45	0xe592...1564	0xd9e1...8b9f
# MEV activities	394,374	229,723	186,143	145,930	137,818
Label	MEV Bot	Uniswap Router	Uniswap Router	Uniswap Router	Sushiswap Router

H THREE STEALTHY ATTACKS IN BUNDLES

- **Cream Finance exploitation.** The root cause is that Cream Finance does not set the reentrancy protection and AMP has a reentrancy vulnerability. Specifically, AMP contains a hook function tokensReceived of ERC777 (https://eips.ethereum.org/EIPS/eip-777) standard, and the hook function can trigger the execution of the attacker's contract. The attacker exploited the reentrancy vulnerability in AMP token contract, and re-loaned assets from Cream Finance. Specifically, the adversary first borrowed AMP from Cream Finance, and then exploited the reentrancy vulnerability in AMP token contract to re-borrow ETH from Cream Finance. Finally, the adversary received profits due to multiple loans with single collateral.

- **xToken exploitation.** The root cause is that xToken mints xSNXa with relying on the price of SNX in Uniswap. The attacker conducted an indirect price manipulation attack [89] on xToken. Specifically, the attacker first borrowed SNX from flash loans, and sold the SNX to Uniswap aiming at tanking SNX's price. Then, the attacker used ETH to mint xSNXa at the tanked price of SNX. Finally, the attacker burned xSNXa and claimed SNX tokens. The attacker paid back the flash loans and exchanged the remaining SNX into ETH as profits.

- **RigoBlock whitehat rescue.** Fig. 21 shows the code snippets of the RigoBlock. The comments at Line 1 and 4 show that the two functions, i.e., setMultipleAllowances and operateOnExchange, should be only invoked by owner. But, the developers miss setting the onlyOwner modifier [27] for the first function although onlyOwner modifier is set for the second function at Line 5. The vulnerability allows non-owners to invoke the first function and set allowances for approved tokens to them. Before the attack transaction, the whitehat invoked the first function and set allowances for them.

Then in the attack transaction, the whitehat drained out six kinds of assets from RigoBlock, and exchanged all of them into ETH. After the attack transaction, the whitehat communicated with RigoBlock in Twitter and returned the ETH to RigoBlock.

I BACKGROUND ON REPRESENTATION LEARNING

Representation learning is a class of machine learning methods to automatically discover the features for constructing classifiers or other predictor variables [35, 67]. Representation learning is fully data-driven and task-oriented, obviating considerable manual efforts for data study and manually extracting features (e.g., feature engineering) [35, 67]. There are two main types of representation learning, namely supervised representation learning and unsupervised representation learning [35]. Supervised representation learning learns features from labeled data, such as neural networks, multi-layer perceptrons, and supervised dictionary learning [35]. Unsupervised representation learning learns features from unlabeled data, such as unsupervised dictionary learning, independent component analysis, automatic coding, matrix factorization [35].

J RETRAINING COST OF OUR MODEL

• **The cost of MEV labeling for bundles**. The MEV activities in bundles can be automatically labeled by summarizing heuristics like existing studies [47, 72, 82] to detect known and discovered MEV activities, and hence the MEV labeling is high-efficient.

• **The computational cost of training the model**. The training of our model only costs a few minutes (< 1 hour) on our server (i.e., Intel Xeon W-1290 CPU with 10 cores at 3.2 GHz).

K COLLECTING WAITING TIMES

We reused methods in [61] to obtain a nine-day waiting time dataset for transactions from Mar. 14, 2023, to Mar. 22, 2023. Specifically, for a transaction *TX*, its waiting time = T^{TX}_{block} - $T^{TX}_{mempool}$, where T^{TX}_{block} is the time when *TX* is mined, and $T^{TX}_{mempool}$ is the time when *TX* first appears in the mempools of miners/validators. With the following methods in [61], we estimate the earliest time when our nodes monitor a transaction as the $T^{TX}_{mempool}$ of this transaction. To estimate the time when transactions first appear as precise as possible, we increase the connectivity of our nodes to the Ethereum P2P network by three methods [61]: i) deploy our nodes to be geographically distributed, ii) configure our nodes with a maximum peer limit of 1,000, and iii) make our nodes to connect to well-known nodes [61]. We further estimate the time when the transaction appeared in a block as the T^{TX}_{block} of this transaction following methods in [61].