TU WIEN Informatics

Towards understanding Multi-Block Maximal Extractable Value in Ethereum

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Jakob Brachmann, BSc
Matrikelnummer 01525526

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.-Prof. Dipl.-Ing. Mag. Dr.techn. Edgar Weippl
Mitwirkung: Mag. Nicholas Stifter, Bakk. techn.

Wien, 2. September 2024

Jakob Brachmann | Edgar Weippl

TU WIEN Informatics

Towards understanding Multi-Block Maximal Extractable Value in Ethereum

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Jakob Brachmann, BSc
Registration Number 01525526

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.-Prof. Dipl.-Ing. Mag. Dr.techn. Edgar Weippl
Assistance: Mag. Nicholas Stifter, Bakk. techn.

Vienna, September 2, 2024

Jakob Brachmann　　　　Edgar Weippl

Erklärung zur Verfassung der Arbeit

Jakob Brachmann, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 2. September 2024

Jakob Brachmann

Acknowledgements

I want to wholeheartedly thank my supervisor, Nicholas Stifter, for providing invaluable feedback, motivating me, and, of course, for the exciting discussions that inspired me during my work.

Kurzfassung

Die Erfindung von Ethereum und seinen Smart-Contract-Fähigkeiten hat den Aufstieg von Decentralized Finance (DeFi) vorangetrieben und ein neues, Blockchain-basiertes Paradigma für Finanzdienstleistungen geschaffen. DeFi-Nutzer stehen jedoch vor einer erheblichen Bedrohung, die monatliche Verluste in Millionenhöhe verursacht: Maximal Extractable Value (MEV).
MEV bezeichnet den Gewinn, der durch die Manipulation der Reihenfolge von Transaktionen innerhalb eines Blocks erzielt wird. In Ethereum ist dies möglich, weil Validatoren in der Lage sind, Blöcke vorzuschlagen und dardurch deren Transaktionsreihenfolge zu bestimmen. Außerdem ist ein Marktplatz entstanden, auf dem Validatoren den Inhalt des ihnen zugewiesenen Blocks verkaufen wodurch professionelle Akteure MEV-Möglichkeiten ausnutzen können.
In den letzten Jahren hat das Problem von MEV in Ethereum zugenommen und damit erhebliches Forschungsinteresse geweckt. Eine Schwierigkeit in der Erforschung von MEV ist allerdings, dass Akteure monetäre Anreize haben, ihre Strategien zu verbergen, da deren Offenlegung zur Nachahmung führen würde. Es wurden sogar Bots beobachtet, welche die in Transaktionen enthaltenen Stragegien nachahmen, um davon zu profitieren, ohne die zugrunde liegenden Strategien zu verstehen.
Obwohl Methoden zur Identifizierung und Quantifizierung von MEV existieren, bleiben Herausforderungen bestehen, insbesondere bei komplexeren Strategien wie Blockchain übergreifendes MEV und Multi-block MEV.
Diese Masterarbeit zielt darauf ab, das Verständnis von Multi-block MEV zu vertiefen. Sie beginnt mit einer umfassenden Übersicht über die aktuelle Literatur zu MEV und identifiziert Ungenauigkeiten und Diskrepanzen in der MEV-Quantifizierung, selbst in Single-Block-Szenarien. Erhebliche Diskrepanzen wurden in den Ergebnissen von MEV Quantifizierungs-Tools wie MEV-Inspect und EigenPhi gefunden, was den Bedarf an zuverlässigeren MEV-Erkennungsmethoden unterstreicht. Um sich diesen Problemen zu widmen, verbessert diese Arbeit die MEV-Erkennungskapazitäten des Open-Source-Tools MEV-Inspect und führt innovative Heuristiken zur Berechnung von Liquidationsgewinnen ein. Auf diesen Verbesserungen aufbauend wird eine Untersuchung von Multi-block MEV durchgeführt. Eine Missed-Block-Strategie wird formuliert und deren theoretische finanzielle Machbarkeit bewertet. Darüber hinaus zeigt die Analyse, dass block builder mehr für aufeinanderfolgende Blöcke zahlen, was darauf hindeutet, dass Multi-block MEV-Strategien bereits eingesetzt werden.

Abstract

The emergence of Ethereum and its Smart Contract capabilities has spurred the rise of Decentralized Finance (DeFi), creating a new, blockchain-based paradigm for financial services. However, DeFi users face a significant threat that results in millions of dollars in monthly losses: Maximal Extractable Value (MEV).

MEV refers to the profit gained by manipulating the order of transactions within a block. In Ethereum, this is possible because validators have the authority to propose blocks and determine their transaction order. Furthermore, a marketplace has emerged where validators sell this block space, allowing professional actors to participate and thus also exploit MEV opportunities.

In recent years, the issue of MEV has become increasingly prevalent in Ethereum, thus attracting significant research attention. This research, however, is complicated by the monetary incentives for players to conceal their strategies, as revealing them would lead to their replication by others. Even bots have been observed copying other MEV-exploiting transactions to profit from them without understanding the underlying strategies. While methods for identifying and quantifying MEV exist, challenges remain, especially with more complex strategies such as cross-chain MEV and Multi-block MEV.

This thesis aims to advance the understanding of Multi-block MEV. It begins with a comprehensive review of the current literature on MEV, identifying major blind spots and discrepancies in MEV quantification, even in single-block settings. Notably, significant discrepancies are found in the results generated by tools designed for MEV detection, such as MEV-Inspect and EigenPhi, underscoring the need for more reliable MEV detection methodologies.

To address these challenges, this thesis enhances the MEV detection capabilities of the open-source tool MEV-Inspect and introduces innovative heuristics for calculating Liquidation profits. Building on these improvements, a preliminary investigation into Multi-block MEV is conducted. A simple Missed-Block strategy is formulated, and its theoretical financial feasibility is assessed. Furthermore, the analysis reveals that block builders pay a premium for consecutive blocks, suggesting that Multi-block MEV strategies are already employed.

Contents

CHAPTER 1

Introduction

In 2008, the person or group behind the pseudonym Satoshi Nakamoto introduced the first blockchain [Nak08]. A blockchain is a permissionless, distributed ledger consisting of blocks containing transactions. Each block is cryptographically linked to the previous block by hash functions to ensure the immutability of the blockchain. Transactions propagate in a peer-to-peer network, and eventually, so-called block proposers propose new blocks to be added on top of the blockchain.

Ethereum is another widespread blockchain that introduced the concept of Smart Contracts [B+14]. Smart contracts are small programs deployed on the Ethereum blockchain that enable various decentralized applications. These Smart Contracts have spurred the rise of Decentralized Finance (DeFi), creating a new, blockchain-based paradigm for financial services. However, DeFi users face a significant threat that results in millions of dollars in monthly losses: Maximal Extractable Value (MEV) [DGK+20, QZG22, Eig24a, Wun23, PJL+23].

MEV refers to the profit gained by manipulating the order of transactions within a block. This manipulation is possible in blockchains because block proposers can choose which transactions to include in a block and their order. Furthermore, a marketplace has emerged where validators sell this block space, allowing professional actors to participate and thus also exploit MEV opportunities.

In recent years, the issue of MEV has become increasingly prevalent in Ethereum, thus attracting significant research attention [DGK+20, OSS+21, YZH+22, HW22, RK20]. This research, however, is complicated by the monetary incentives for players to conceal their strategies, as revealing them would lead to their replication by others. Even bots have been observed copying other MEV-exploiting transactions to profit from them without understanding the underlying strategies.
Due to the specific Proof of Stake (PoS) implementation in Ethereum, the validators

allowed to propose a block in a slot are known two [1] epochs ahead of time [2]. This predictability of when a validator is allowed to propose a slot simplifies the process of outsourcing block-building to specialized block builders. This outsourcing is called Proposer-builder separation (PBS). Block builders create blocks with transactions they pick up through the peer-to-peer network and their optional MEV exploiting transactions. They then offer these blocks to the validators, together with a fee. In the most distributed PBS implementation MEV-Boost [3], block builders submit their blocks to relays, which validate the block content. After validation, the relays submit the block's header without its content and the offered fee to the validator, who then chooses the most profitable block. After the validator cryptographically signs and commits to publishing the block, the relay sends the actual block content to the validator.
This mechanism opens the theoretical opportunity for Multi-block MEV (MMEV). MMEV is MEV that is extracted by applying strategies that span multiple blocks. PBS simplifies MMEV, as MEV extractors, often called *MEV searchers*, can buy consecutive blocks on the marketplace. While single-block MEV is the focus of current research, little literature exists on MMEV, and it is unclear if it is currently being actively deployed.

This thesis aims to advance the understanding of Multi-block MEV. As a foundation for this research, this thesis offers a comprehensive review of the current literature on MEV and a state-of-the-art analysis of Single-block MEV detection.
Single-block MEV detection is not trivial because MEV exploiters employ increasingly complex strategies. As mentioned, these actors have a monetary incentive to conceal their strategies, as these are at risk of being copied, decreasing their rentability.

Studying MEV is motivated by the high monetary losses it produces for regular DeFi users [Noy21]. In addition, MEV boosts the centralization of block-building and generally increases distrust in blockchain technology.
Since MEV is a popular topic in research, awareness of its dangers has risen over the last few years. More and more decentralized applications and Smart Contract developers aim to develop MEV-resistant technology.
The topic of MMEV, however, is currently not discussed broadly in the scientific and developer community. This attack vector is, therefore, often not considered in the design phase of decentralized applications. Successful MMEV attacks could, however, pose a significant danger to decentralized applications' integrity and could lead to significant financial losses for users. Therefore, it is important to understand the MMEV landscape, its concrete strategies, and how it is already in use today.

[1] At the end of epoch N_t, the validators for epoch N_{t+2} are chosen. See https://ethresear.ch/t/selfish-mixing-and-randao-manipulation/16081

[2] One epoch consists of 32 slots, each 12 seconds long. So, one epoch is roughly 6.4 minutes in total. https://info.etherscan.com/epoch-in-ethereum/

[3] https://github.com/flashbots/mev-boost

1.1 Objectives & Research Questions

This thesis aims to investigate the nature of Multi-block Maximal Extractable Value attacks within the Ethereum blockchain ecosystem. To build a comprehensive understanding of Multi-block MEV attacks, exploring the MEV landscape's current state is essential. This leads to the formulation of the first research question:

> *RQ_1: What is the current state-of-the-art in MEV attacks, and what mitigation strategies have been implemented or proposed?*

Permissionless blockchains need to be transparent to allow for decentralized and public validation. Therefore transactions and account balances are openly available. However, MEV exploiting techniques can be very complex, especially if they involve multiple transactions, dedicated MEV exploiting Smart Contracts, or transactions involving multiple blockchains. Therefore, to confidently make statements about the nature of MMEV, one first has to understand how MEV can be detected. Therefore, the next research question aims to understand the current state-of-the-art MEV detection algorithms and techniques.

> *RQ_2: What are the state-of-the-art methods for detecting and quantifying Single-block Maximal Extractable Value in Ethereum, and what are the limitations of these methods?*

In exploring Multi-block MEV attacks, it is crucial to understand how an MEV searcher could potentially exert control over a sequence of consecutive blocks. This leads to our third research question:

> *RQ_3: How can an MEV searcher gain control over consecutive sequences of blocks, and are there any indications that these methods are currently being employed?*

Finally, to understand Multi-block MEV attacks, we explore the economic feasibility of a simple form of a Multi-block MEV attack:

> *RQ_4: Is the value of MEV in the block following a missed slot statistically higher, and would this make an attack where the attacker intentionally misses a block economically viable?*

1.2 Outline

1.2.1 Methodology

The following two methodological approaches were chosen to investigate the research questions.

1. Literature Review
   This review will focus on understanding the current MEV landscape, including the various forms of MEV attacks, multiple dimensions of MEV, the negative externalities of MEV, and mitigation mechanisms. Special focus is given to analyzing current state-of-the-art MEV detection and quantification mechanisms. To ensure transparency and reproducibility in the literature selection process, we will conduct a systematic literature review by adhering to the PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) [4] guidelines.

2. Empirical Evaluation
   A comprehensive analysis of blockchain data is required to determine whether Multi-block Maximal Extractable Value (MMEV) strategies are actively employed or if there are theoretical incentives for their deployment. This involves examining on-chain data using a custom Ethereum beacon node to gain insights into MMEV activities.

   In addition to on-chain data analysis, various MEV analysis tools are assessed to evaluate their effectiveness. The outputs generated by these tools for identical data ranges are compared through statistical analyses to identify discrepancies and measure their reliability. Furthermore, the source code of these tools is analyzed to uncover the primary factors contributing to variations in results.

   To investigate methods for securing a consecutive block sequence, we first describe two approaches an MEV searcher might use:

   - Participating as a validator in a staking pool
   - Purchasing a sequence through a Proposer-builder separation (PBS) marketplace

   We calculate the probability of a staking pool acquiring a consecutive sequence of blocks and analyze on-chain data to detect such occurrences. For the PBS marketplace approach, we analyze both on-chain Ethereum data and published data from PBS marketplaces to study the acquisition of consecutive sequences.

   Finally, we identify missed slots using a state-of-the-art Ethereum beacon node to statistically analyze whether a block's MEV value increases following a missed slot.

[4]https://www.prisma-statement.org/

This MEV value is calculated using advanced MEV detection tools to provide a quantitative assessment of changes in economic value after missed slots.

1.2.2 Structure of Work

1. **Introduction** This chapter outlines the motivation behind this thesis and presents the addressed research questions.

2. **Technical Background** This chapter provides the technical foundation for this thesis. It covers blockchain fundamentals and offers an overview of Ethereum, focusing on its Proof of Stake system and Decentralized Finance (DeFi) applications.

3. **MEV - Maximal Extractable Value**
   This chapter reviews the current state of the art in MEV research, detailing the different types of MEV attacks and characterizing various proposed and implemented mitigation strategies.

4. **A systematic literature review of state-of-the-art MEV detection** This chapter conducts a systematic literature review to evaluate the state-of-the-art MEV detection and quantification methods. The review identifies and analyzes 16 papers and tools, comparing and classifying their MEV detection mechanisms. Additionally, this chapter discusses the advantages and limitations of these techniques.

5. **State-of-the-art MEV detection** This chapter further explores state-of-the-art MEV detection methods by examining the output of selected detection tools. Based on this analysis, a new heuristic for detecting Liquidations is introduced. The chapter also discusses the challenges of MEV detection and the shortcomings of existing approaches.

6. **Multi-Block MEV** This chapter investigates Multi-block MEV strategies, describing two techniques an MEV searcher might use to secure a Multi-block sequence. It analyzes these techniques using real on-chain data and introduces a simple Multi-block MEV technique, assessing its economic feasibility.

7. **Conclusion** The final chapter summarizes the research conducted, reiterates the research questions, and presents the thesis results. It also provides insights into potential future work on this topic.

CHAPTER 2

Technical Background

This chapter provides the technical foundation crucial to this thesis. We begin by exploring the fundamental principles of blockchain technology, using Bitcoin—the first and most well-known blockchain—as an illustrative example. Following this, we delve into Ethereum, the blockchain at the core of our study, offering detailed insights into its architecture and functionalities.

Key topics include Smart Contracts, the Proof of Stake (PoS) mechanism, and Decentralized Finance (DeFi), all of which are vital for understanding Maximal Extractable Value (MEV).

2.1 Bitcoin

In 2008, the popular blockchain Bitcoin was introduced by a person or group behind the pseudonym Satoshi Nakamoto [Nak08]. The aim was to create a digital currency that could be exchanged without involving a third party. Such an exchange, a transaction, transfers a specific amount of Bitcoin from one user to another. Each transaction is cryptographically signed to ensure that only the coin's owner can transfer it.

However, signatures alone cannot prevent double-spending, where a user attempts to spend the same coin multiple times. To counter this, the decentralized system requires a unified view of the transaction order. This common view is established by the blockchain. A blockchain is a distributed ledger where each block contains an ordered set of transactions, defining the correct sequence of events. These blocks are linked through cryptographic hashes, with each block containing the hash of the previous one. This linkage ensures that any tampering with a block would alter the hashes of all subsequent blocks, making the manipulation detectable. An illustration of this linkage is seen in Figure 2.1.

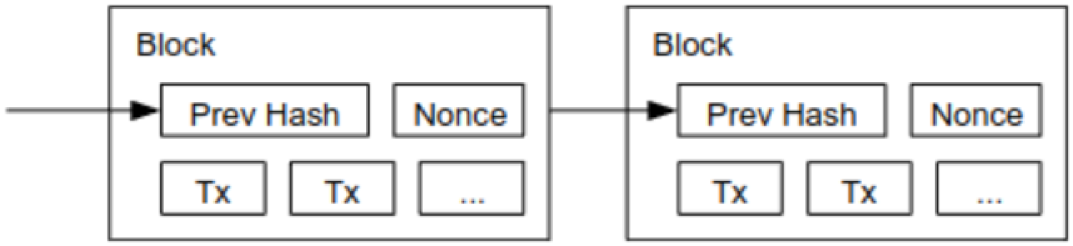


Figure 2.1: A simple illustration of the Bitcoin blockchain. Each block is linked to the previous block by containing its hash. The Nonce field is set in the proof-of-work proof, which secures the blockchain against manipulation attacks. [Nak08]

Deciding globally which transactions are included in a block and their order presents a significant challenge. A simple voting system could be skewed by a single entity controlling multiple identities, disproportionately influencing the vote. This type of attack, where an entity mimics multiple entities, is known as a Sybil attack [Dou02].

Bitcoin itself addresses the double-spending problem with a proof-of-work mechanism. The mechanism operates as follows: Each block contains a Nonce field. To publish a block, a miner must find a nonce value such that the block's hash is lower than a specified difficulty threshold. Since this task cannot be solved efficiently through an algorithm [1], miners must use brute force to test multiple nonce values until they find one that meets the criteria. This process requires significant computational effort, hence the term "proof-of-work."

The difficulty level, which determines the validity of the proof-of-work, also regulates the interval between block publications. If an increase in overall computing power reduces the average time it takes the network to publish a new valid block, the Bitcoin protocol adjusts the difficulty to maintain the target interval of 10 minutes between blocks. This adjustment takes place every 2016 blocks.

This proof-of-work mechanism also enhances the blockchain's resistance to manipulation. Altering a block's content would necessitate rehashing it and finding a new nonce that satisfies the difficulty condition. Consequently, the computational work must be redone. Since subsequent blocks are linked through the previous block's hash, manipulating one block would require redoing the nonce finding for all following blocks.

Miners are incentivized to spend the required computational power through the Block Reward, a specified number of coins awarded to the miner who successfully publishes a valid block with a valid proof-of-work. This reward motivates many miners to work on solving the proof-of-work puzzle simultaneously. As a result, two or more blocks can

[1]An efficient solution would compromise the desirable properties of the cryptographic hash function, especially preimage resistance, which is the property that makes it computationally infeasible to reverse-engineer the original input from its hash output.

occasionally be published at nearly the same time, all building on the previous block. Depending on the network topology, different network parts might continue building on different blocks, creating a *fork*. When a node, an entity running the Bitcoin software, encounters two forks, it applies a Fork Choice Rule. For Bitcoin, this rule is to accept the longer chain with more invested proof-of-work. The part of the network with the majority of computational power will eventually produce a longer chain. With the assumption that most nodes are *honest nodes*, which are nodes that correctly follow the Bitcoin protocol, all honest nodes will share a common prefix of blocks in their respective blockchains [GKL24]. This means that while individual blockchains may differ in the most recent blocks, the sequence of blocks leading up to that point will be identical across all honest nodes.

2.2 Ethereum

This thesis focuses on the Ethereum blockchain, introduced in 2014 by Buterin et al. [B+14]. While Ethereum shares fundamental principles with Bitcoin, it markets itself as an enhanced version, offering additional functionality.

One key feature is its built-in Turing-complete programming language, which enables the creation of small programs known as Smart Contracts. This programming capability has spurred the development of various applications that benefit from the blockchain's security properties and the Ethereum community's decentralization efforts.

Ethereum's internal currency is Ether, abbreviated as ETH. The smallest unit of Ether is called Wei, equivalent to 10^{-18} Ether.

The following section explains Ethereum's technical concepts, including addresses, the Ethereum Virtual Machine, and Smart Contracts. We will then discuss an application of Ethereum: Decentralized Finance (DeFi).

2.2.1 Addresses

Ethereum utilizes the concept of Accounts, identified by addresses (160-bit identifiers) [Eth24a]. There are two types of accounts: Externally Owned Accounts (EOAs) and Contract Accounts.

EOAs are created by generating a public/secret key pair, with the public address derived from the hash of the public key. The owner of the secret key controls the corresponding address, which allows them to send Ether to other EOAs or Contract Accounts or trigger functions in Contract Accounts.
In contrast, Contract Accounts are governed not by secret keys but by the logic of their underlying contract code. Smart Contracts are established through a contract creation transaction, which deploys the contract code. Like EOAs, Contract Accounts can transfer Ether, call functions in other Contract Accounts, and even create new Contract Accounts. However, Contract Accounts can only perform actions when triggered by an EOA or

another Contract Account via a transaction. As a result, autonomous cron-job-like functionality is impossible for Contract Accounts.

2.2.2 Transactions

Generally, a valid transaction can be seen as a transition from one state to another, affecting elements such as account balances, Smart Contracts, and contract storages [W+14]. A transaction can modify this state in three different ways:

1. **Message Call**
   This involves invoking a Smart Contract function, which can be initiated by an EOA, another Smart Contract, or even a Smart Contract calling its functions. The behavior and side effects are determined by the Smart Contract code.

2. **Contract Creation**
   This deploys Smart Contract bytecode to the blockchain.

3. **Value Transfer**
   A value transfer is transferring Ether between two EOAs, or one EOA and a Smart Contract.

All transactions consume gas, representing the computational effort required and paid by the transaction sender in Ether. Each EVM instruction (see subsection 2.2.4) consumes a predefined amount of gas [2]. The gas concept was introduced to prevent network spamming or denial-of-service attacks by making such attacks economically unviable due to high transaction fees. It also prevents accidental or intentional infinite loops by allowing the transaction sender to limit the gas consumed and setting an overall gas limit per block, currently 30,000,000 gas [3].

The transaction fee is calculated as *gas used * (base fee + priority fee)*. The base fee, determined by the protocol, is calculated based on previous blocks and is burned when the transaction is included in a block, providing no incentive for block proposers to include transactions that only pay the base fee. The sender can pay a tip by increasing the priority fee to increase the likelihood of inclusion [Eth24b].

A transaction is a *failed transaction* if it is cryptographically signed by the sender but violates other constraints, such as insufficient funds or a Smart Contract assertion violation. The blockchain still includes failed transactions, and the sender must pay the gas fee, but these transactions do not alter the blockchain state.

A transaction consists of the following components:

[2]https://ethereum.org/en/developers/docs/evm/opcodes/
[3]https://etherscan.io/blocks

- **from**
  The address of the transaction sender must be an EOA since only EOAs can trigger transactions.
- **to**
  The receiver of the transaction. If the receiver is an EOA, it is a value transfer; if it is a contract account, it is a message transaction.
- **signature**
  The transaction's cryptographic signature must be signed by the sender's secret key corresponding to the "from" address.
- **nonce**
  The nonce is a counter for the transaction the sender's account sends. Including the same transaction with the same nonce twice is prohibited, which prevents replay attacks. In a replay attack, an attacker could attempt to republish a transaction to receive the same amount of Ether again. For example, if address A sends 1 Ether to address B, B could try to publish the same transaction again after it has been included in the blockchain, attempting to receive 2 Ether. However, the second transaction would be invalid due to the duplicate nonce.
- **value**
  The amount of Ether sent from the sender to the recipient (denominated in Wei).
- **input data**
  An optional field containing arbitrary data, such as encoded function names and parameter values for contract calls or contract code for contract creation.
- **gasLimit**
  The maximum amount of gas the transaction can consume.
- **maxPriorityFeePerGas**
  The maximum priority fee (tip) paid to the block proposer.
- **maxFeePerGas**
  The maximum fee the transaction can consume, including the base and priority fees multiplied by the gas consumed.

2.2.3 Transaction propagation and MemPool

As previously mentioned, transactions are valid only when they contain a signature made with the sender's secret key. A state change initiated by a transaction is considered publicly accepted once this transaction is included in a valid block of the commonly accepted blockchain.
A transaction must be received by the block proposer to be included in a block. The general method for disseminating transactions is to use a P2P gossip protocol through

which the transaction is broadcast. The transaction sender sends the signed transaction to a connected Ethereum node. An Ethereum node is a computer running an Ethereum client (see subsection 2.2.7). Each node validates incoming transactions by, for example, verifying the signature against the sender's public key and ensuring the sender has sufficient funds. Invalid transactions are discarded, while valid transactions are broadcast to all known nodes. This process continues until the transaction reaches a node controlled by a block proposer, who includes it in a block and broadcasts it across the network.

When a node receives a valid block, it deletes any transactions included in that block from its locally stored unconfirmed transactions. From this point on, the transaction cannot be included in a future block since a nonce cannot be reused for the same account (see subsection 2.2.2). The global view of all signed transactions not yet included in a block is called the MemPool. Due to the asynchronous nature of transaction propagation among Ethereum nodes, not all nodes maintain the same view of the global MemPool.
A Smart Contract transaction contains the decoded function signature in the input data field. Since transactions are only signed and not encrypted, their information is public, allowing all participating nodes to derive the sender's intent. Bad actors can exploit this transparency for their gain, often to the disadvantage of the original sender [4].
To address this issue, private MemPools have emerged, providing direct channels between transaction senders and block proposers. According to a dashboard by Toni Wahrstätter [5], from June 3, 2024, to July 3, 2024, approximately 9.55 % of all transactions were private.

2.2.4 Ethereum Virtual Machine

The Ethereum Virtual Machine (EVM) is responsible for executing Smart Contracts and is a fundamental component of the Ethereum network. It deterministically specifies how a transaction modifies the Ethereum world state, producing a new state from a given old state and a set of valid transactions [W+14].

The EVM is a stack-based, 256-word machine with a stack size of 1024.
When a transaction triggers a Smart Contract's function, the byte code of this Smart Contract is used as input for the EVM's execution. For this execution, the EVM uses volatile storage as a word array.

Although the EVM is Turing complete, it is often described as *quasi*-Turing complete [W+14] because it does not support infinite loops in practice. While it is possible to create infinite loops through EVM instructions like JUMP or JUMPI or through recursive function calls within or across contracts, the gas limit effectively constricts these loops. The gas limit imposes a cap on the number of instructions that can be executed, preventing indefinite execution and ensuring that computations are finite [B+14].

[4]This problem is covered in more detail in chapter 3
[5]`https://mempool.pics/` Accessed on 2024-07-03

2.2.5 Smart Contracts

Smart Contracts are a key feature that distinguishes Ethereum from Bitcoin. They are deployed through contract creation transactions, and each Smart Contract is uniquely identified by its contract address. Once deployed, Smart Contracts offer interfaces that can be accessed by other transactions or contracts. They can modify the blockchain state by reading from or writing to storage or invoking other Smart Contracts.

Smart Contracts can manage Ether similarly to EOAs, but unlike EOAs, they do not possess a secret key and, therefore, cannot directly sign transactions. Consequently, sending Ether from a Smart Contract must be explicitly implemented in the contract's logic and can only be triggered by a transaction initiated from an EOA. As a result, Smart Contracts typically incorporate their own authentication and permission mechanisms.

The code for a Smart Contract, deployed through a contract creation transaction, is written in Ethereum Virtual Machine (EVM) bytecode. Several high-level programming languages have been created to facilitate Smart Contract development, including Solidity [6], Vyper [7], and Cairo [8]. Among these, Solidity is the most widely used [9].

In the following section, we will examine the functionality of Solidity through the example of a Smart Contract deployed at address 0xba1...bf2c8 [10]. For clarity and brevity, the Solidity code has been pruned and modified. We will focus on key features relevant to the blockchain environment, including security.

The contract (Listing 2.1) is an implementation of a flash loan. A flash loan is a type of loan that must be repaid within the same transaction. This means that tokens can be borrowed temporarily, but if they are not returned by the end of the transaction, the entire transaction is reversed. Flash loans are commonly utilized by MEV searchers because they minimize the need for locked capital.

```
// SPDX-License-Identifier: GPL-3.0-or-later
pragma solidity ^0.7.0;
pragma experimental ABIEncoderV2;

import "../lib/helpers/BalancerErrors.sol";
import "../lib/openzeppelin/IERC20.sol";
import "../lib/openzeppelin/ReentrancyGuard.sol";
import "../lib/openzeppelin/SafeERC20.sol";

import "./Fees.sol";
import "./interfaces/IFlashLoanRecipient.sol";


abstract contract FlashLoans is Fees, ReentrancyGuard, TemporarilyPausable {
    using SafeERC20 for IERC20;
```

[6]https://soliditylang.org/ Accessed on 2024-07-03
[7]https://docs.vyperlang.org/en/stable/ Accessed on 2024-07-03
[8]https://www.cairo-lang.org/ Accessed on 2024-07-03
[9]https://defillama.com/languages Accessed on 2024-07-03
[10]https://etherscan.io/address/0xba12222222228d8ba445958a75a0704d566bf2c8#code

```solidity

    function flashLoan(
        IFlashLoanRecipient recipient,
        IERC20[] memory tokens,
        uint256[] memory amounts,
        bytes memory userData
    ) external override nonReentrant whenNotPaused {
        InputHelpers.ensureInputLengthMatch(tokens.length, amounts.length);

        uint256[] memory feeAmounts = new uint256[](tokens.length);
        uint256[] memory preLoanBalances = new uint256[](tokens.length);

        // Used to ensure `tokens` is sorted in ascending order, which ensures token uniqueness.
        IERC20 previousToken = IERC20(0);

        for (uint256 i = 0; i < tokens.length; ++i) {
            IERC20 token = tokens[i];
            uint256 amount = amounts[i];

            require(token > previousToken, token == IERC20(0) ? "ZERO_TOKEN" : "UNSORTED_TOKENS");
            previousToken = token;

            preLoanBalances[i] = token.balanceOf(address(this));
            feeAmounts[i] = _calculateFlashLoanFeeAmount(amount);

            require(preLoanBalances[i] >= amount, "INSUFFICIENT_FLASH_LOAN_BALANCE");
            token.safeTransfer(address(recipient), amount);
        }

        recipient.receiveFlashLoan(tokens, amounts, feeAmounts, userData);

        for (uint256 i = 0; i < tokens.length; ++i) {
            IERC20 token = tokens[i];
            uint256 preLoanBalance = preLoanBalances[i];

            // Checking for loan repayment first (without accounting for fees) makes for simpler debugging, and results
            // in more accurate revert reasons if the flash loan protocol fee percentage is zero.
            uint256 postLoanBalance = token.balanceOf(address(this));
            require(postLoanBalance >= preLoanBalance, "INVALID_POST_LOAN_BALANCE");

            // No need for checked arithmetic since we know the loan was fully repaid.
            uint256 receivedFeeAmount = postLoanBalance - preLoanBalance;
            require(receivedFeeAmount >= feeAmounts[i], "INSUFFICIENT_FLASH_LOAN_FEE_AMOUNT");

            _payFeeAmount(token, receivedFeeAmount);
```

```
61                  emit FlashLoan(recipient, token, amounts[i], receivedFeeAmount);
62              }
63          }
64      }
```

Listing 2.1: Example Smart Contract representing a Flash Loan written in Solidity

The logic of this contract in Listing 2.1 revolves around a function called *flashLoan*. Users who invoke this function specify the tokens and amounts they wish to borrow. The contract first verifies that it has adequate funds to cover the loan (lines 41 and 42) and calculates the fee associated with each loaned token (line 39). Subsequently, the requested funds are sent to the recipient.

In line 45, the contract calls a function on the recipient, which must be a Smart Contract with a *receiveFlashLoan* callback function. This function allows the recipient to utilize the borrowed tokens in any way they see fit. For instance, MEV searchers might leverage these tokens to exploit arbitrage opportunities by buying and selling other tokens. After the recipient's Smart Contract completes its operations, control returns to the flash loan contract, which then proceeds to the second part of its logic. At this stage, the borrower must have repaid the entire loan amount along with any fees. The contract checks the repayment and fee payment in this second phase to ensure correctness.

In the following, we will highlight the specifics of Solidity and its Smart Contract development.

Compiler instructions

Lines 2 and 3 of the contract in Listing 2.1 specify compiler directives. Line 2 sets the minimum required version of the Solidity compiler, while Line 3 enables an experimental feature. This feature was experimental until version v0.8.0 and is part of the compiler in higher versions [11].

Inheritance

As an object-oriented programming language, Solidity supports inheritance, allowing contracts to be derived from one or more base contracts. In this example (Line 14), the contract inherits from *Fees*, *ReentrancyGuard*, and *TemporarilyPausable*, demonstrating how inheritance facilitates code reuse and separation of concerns.

Modifiers

Modifiers in Solidity function similarly to aspects in aspect-oriented programming. They are executed before the body of a function and can alter the function's behavior. Modifiers are commonly used to enforce preconditions or guards for functions, ensuring that certain conditions are met before the function executes. For instance, in the example contract,

[11] https://docs.soliditylang.org/en/v0.8.0/080-breaking-changes.html

the *flashLoan()* function is protected by two modifiers: *nonReentrant* and *whenNotPaused* (Line 22). These modifiers help prevent reentrancy attacks and ensure the contract is not paused before the function proceeds.

```
    /**
     * @dev Reverts if the contract is paused.
     */
    modifier whenNotPaused() {
        require(_isNotPaused(), Errors.PAUSED);
        _;
    }
```

Listing 2.2: Example of a modifier in Solidity

Listing 2.2 presents a simplified definition of the whenNotPaused modifier. By calling the *_isNotPaused()* function, this modifier ensures the contract is not paused. The _; syntax in the modifier indicates where the body of the modified function will be inserted and executed.

Key Solidity Features for Smart Contract Development

Solidity is a specialized programming language designed to develop Smart Contracts on the Ethereum platform. It offers features specifically crafted to meet the unique requirements of Smart Contract development. The following will present the most important features based on the *flashLoan* example.

require()

The *require()* statement helps ensure the integrity and security of a Smart Contract. It allows the definition of conditions, e.g., invariants, that must be met before the code execution can continue. It takes as an argument a boolean condition that must be evaluated to be true for the code execution to continue. If this condition fails, then the transaction gets reverted, which results in the reversion of all state changes caused by the transaction.

Line 58 of the example contract shows such a *require()* usage. It checks that the value of the variable *postLoanBalance* must be greater or equal to the value of *preLoanBalance*. This ensures that the flash loan lender has repaid the loan. If this condition is evaluated as false, then this transaction and its state changes will be reverted, and an exception with the passed message will be thrown.

Address Type and Ether Transfer

Solidity includes a built-in address type representing an Ethereum address, which can be either an externally owned account (EOA) or a Smart Contract address. Additionally, Solidity offers an address payable type specifically for handling addresses that can receive Ether, providing convenient functions like *transfer* and *send* for sending Ether [12].

[12]https://docs.soliditylang.org/en/latest/types.html#address

Events

Solidity enables Smart Contracts to emit events [13], which serve as a form of logging. These logs are stored by the Ethereum Virtual Machine (EVM) and can be subscribed to by external applications through an Ethereum client. This allows them to be notified about state changes in decentralized applications. Line 61 illustrates an example of such an event emission, notifying all listeners that a flash loan has been executed.

Fallback and Receive Functions

Solidity allows the definition of special [14] functions specifically tailored for Smart Contract development. One such function is the *receive* function, called when Ether is transferred to the Smart Contract without any accompanying data (calldata). This indicates a plain Ether transfer, not a function call.

Another special function is the *fallback* function. This function is invoked when no receive function is defined for handling plain Ether transfers. Additionally, it is called if no other function matches the provided function signature.

This and Self-Destruct

Solidity provides a way for a smart contract to reference itself through the *this* keyword. This can be utilized to check the contract's Ether balance. One can retrieve the Smart Contract balance in Wei by calling *address(this).balance*.

Another key Solidity feature is the *selfdestruct* function. This function enables a contract to remove itself from the blockchain and transfer its remaining Ether to a designated address.

2.2.6 Proof of Stake

Ethereum initially employed a Proof of Work (PoW) consensus mechanism. In 2022, it transitioned to Proof of Stake (PoS) through an upgrade known as "The Merge." According to the Ethereum Foundation, this transition reduced the energy consumption of the Ethereum network by 99.95% [15].

In the following, we will present the building blocks of Proof of Stake Ethereum.

Validators

The primary participants in the Ethereum PoS mechanism are validators. To become a validator, an individual must deposit 32 ETH into a deposit contract. Validators are responsible for validating and proposing new blocks.

Ethereum time is divided into epochs, each comprising 32 slots. Slots occur at 12-second intervals, with each slot capable of containing one block or remaining empty. The

[13]https://docs.soliditylang.org/en/latest/contracts.html#events
[14]https://docs.soliditylang.org/en/latest/contracts.html#special-functions
[15]https://ethereum.org/en/roadmap/merge/

RANDAO algorithm, which generates pseudo-random values in a decentralized setting, determines which validators can propose a block in each slot.

When a validator's assigned slot arrives, it aggregates transactions into a bundle, known as the execution payload. Validators may prioritize transactions by gas price to maximize profits or employ other MEV strategies (see chapter 3). A valid Ethereum block consists of the execution payload and additional consensus information.

Attestations and Committees

Attestations are a critical part of the PoS consensus mechanism. They are signatures from validators confirming the validity of a block. To manage the attestation process efficiently, validators are divided into committees, with each committee responsible for attesting to a specific slot within an epoch. Each validator is expected to vote once per epoch.

To reduce network congestion, 16 validators from each committee are chosen as aggregators. Validators send their attestations to the aggregators, who combine them into a final aggregation in the block.

Incentives and Penalties

Validators are financially motivated by rewards for participating in the consensus process, such as signing attestations. Block proposers also receive priority fees from transactions within their blocks and possible MEV revenue.

Two primary mechanisms ensure validators behave correctly: their stake and slashing penalties. By staking 32 ETH, validators have a financial incentive to maintain the network's integrity. Any attack that undermines Ethereum would also decrease the value of their staked ETH.

Slashing penalties are enforced for incorrect behavior, such as proposing two different blocks, attesting to multiple blocks in the same slot, or attesting to a block that contradicts a finalized block. Slashed validators are automatically removed from the validator set and incur a reduction in their stake.

Finality and fork choice rule

Finality in Ethereum is managed by the Gasper protocol [BG17]. Finality is the point at which a transaction is considered immutable. In Bitcoin, there technically exists no point at which a block is immutable, but the probability of the block being forked drops with each following block, which is why in practice after six blocks, a block is considered safe. Ethereum uses a voting mechanism to determine finality. A block is justified if two-thirds of validators vote for it at the end of an epoch. If the subsequent epoch is also justified, the block is considered finalized.

Ethereum's fork choice rule prioritizes the chain with the most attestations over chains with fewer attestations, contrasting with Bitcoin's preference for the longest chain. This mechanism ensures that the network converges on a single, agreed-upon history, enhancing the blockchain's security and reliability.

2.2.7 Ethereum Client Implementations

An Ethereum node is a computer that participates in the Ethereum network. To minimize complexity and facilitate parallel development, the functionality of a node is divided into multiple client types.

Firstly, an Ethereum node runs an *execution client*. This client accepts incoming transactions, executes and verifies them within a local Ethereum Virtual Machine (EVM) instance, and stores the current EVM state.

Secondly, an Ethereum node must run a *consensus client* or *Beacon node*. This client synchronizes with the blockchain by receiving new blocks, validating them with the execution client's help, and propagating the known blocks to peers. The consensus client also applies the fork choice rule to resolve any forks in the blockchain.

If the node is operated by a validator, it also runs a *validator client*. This software proposes and attests to blocks, leveraging the consensus and execution of clients to perform its duties.

Since Ethereum's inception, many such clients have been developed. The most popular execution client is *Geth* [16], while the most popular consensus client is *Prysm* [17]. It is crucial for the network's resilience that diverse clients are used to avoid single points of failure. If a client with over 1/3 of the market share encounters a consensus bug, it will prevent the network from reaching finality, which requires a 2/3 majority. Nodes running the faulty client would face significant costs due to an inactivity leak mechanism that slashes the stakes of non-participating validators until finality is restored.

In the worst-case scenario, a corrupted client with a 2/3 majority could finalize invalid blocks [@jm22].
These risks highlight the importance of client diversity.

According to clientdiversity.org, as of July 13, 2024, execution and consensus clients exhibit an unhealthy distribution. Prysm, the leading consensus client, holds a 36.44% market share, while the most popular execution client has a 55% market share. Promoting client diversity is essential to mitigate the risks associated with such concentrations.

2.2.8 Decentralized Finance in Ethereum

One of Ethereum's primary functionalities is sending Ether. However, introducing Smart Contract functionality has enabled so-called decentralized applications (dApps). These dApps encompass various domains, from games and lotteries to voting contracts and an extensive ecosystem of Decentralized Finance (DeFi). In the following section, we will introduce the DeFi ecosystem, as this thesis specifically addresses MEV in the context of DeFi. We will start by introducing tokens, which are crucial tradable assets. Following

[16]https://geth.ethereum.org/
[17]https://github.com/prysmaticlabs/prysm

that, we will delve into the core of DeFi by exploring key concepts such as Automated Market Makers and Lending Protocols.

Tokens

Tokens are digital assets implemented as Smart Contracts on the Ethereum blockchain. They inherit the blockchain's security guarantees, which protect transactions from tampering and ensure that the Token rules and methods for proving ownership, which are defined within the smart contract code, are upheld. The two most important token standards are ERC20 and ERC721. ERC20 tokens are fungible, and ERC721 tokens are non-fungible, meaning each ERC721 token is unique.

ERC20-Tokens

ERC20 [18] tokens are fungible, meaning that each token is identical to another. The ERC20 token standard defines a set of functions and events each implementing token must support.

The first four functions are administrative:

- *name()* Returns the name of the token.
- *symbol()* Returns the token's symbol.
- *decimals()* Returns the number of decimal places the token uses.
- *totalSupply()* Returns the total supply of tokens.

The subsequent functions are essential for managing the token assets:

- *balanceOf(address _owner)* Allows querying the balance of ERC20 tokens for a given address.
- *transfer(address _to, uint256 _value)* Allows a token holder to transfer tokens to another address.
- *transferFrom(address _from, address _to, uint256 _value)* Allows transferring tokens on behalf of another address, which is useful for delegated transactions.

The last function demonstrates an interesting aspect: the ability to send tokens on behalf of another user. This capability is facilitated by the approve and allowance functions. The *approve(address _spender, uint256 _value)* function allows a token holder to approve another address to spend up to a specified number of tokens. The *allowance(address _owner, address _spender)* function enables the spender to check how many tokens they can transfer on behalf of the owner.

[18] https://eips.ethereum.org/EIPS/eip-20

ERC721-Token

ERC721 [19] is the standard for Non-Fungible Tokens (NFTs). Unlike ERC20 tokens, each ERC721 token is unique. The uniqueness is represented by a tokenId, a unique identifier for each token under a given contract. Thus, the Smart Contract's address and tokenId combination is globally unique. Similar to the ERC20 standard, the ERC721 standard defines an interface for transferring and approving token transfers:

- *balanceOf(address _owner)* Returns the number of NFTs owned by _owner.
- *ownerOf(uint256 _tokenId)* Returns the owner of the NFT specified by tokenId.
- *transferFrom(address _from, address _to, uint256 _tokenId)* Transfers ownership of the NFT from one address to another.
- *approve(address _approved, uint256 _tokenId)* Approves another address to transfer the given NFT.

These functions provide a standardized method for managing unique digital assets, making ERC721 tokens ideal for applications such as digital collectibles, houses, and other unique assets.

Key ERC20 Tokens

This section will introduce two crucial ERC-20 tokens in the DeFi landscape: WETH and USDC. While USDC is an example of a stablecoin, WETH serves a different but crucial role.

Wrapped Ether (WETH) The ERC-20 standard provides a unified interface for trading and exchanging tokens on the Ethereum network. However, Ethereum's native currency, Ether (ETH), does not natively adhere to the ERC-20 standard. To address this, WETH (Wrapped Ether) was created. WETH is essentially a wrapper around ETH that conforms to the ERC-20 standard.

The WETH Smart Contract allows users to deposit ETH and receive an equivalent amount of WETH in return. This conversion ensures that one WETH is always valued like one ETH. Conversely, users can also deposit WETH into the contract and receive ETH back. This functionality is crucial for integrating Ether into DeFi protocols that require ERC-20 compatibility.

USDC Another notable token is USDC, which exemplifies the concept of stablecoins. One of the original goals of blockchain technology was to replace traditional currencies. However, since their introduction, cryptocurrencies have often exhibited higher volatility than traditional currencies. This volatility is problematic for their use as a stable means

[19]https://eips.ethereum.org/EIPS/eip-721

of exchange, as the value can fluctuate even between the selection of a product and the payment.

Stablecoins aim to address this issue by maintaining a value close to that of traditional currencies. USDC is a type of stablecoin designed to closely mirror the value of the US Dollar (USD). It achieves this stability by being fully backed by real USD held in reserve. Each USDC token is backed 1:1 by a corresponding USD held by trusted financial institutions. This ensures that one USDC can be exchanged for exactly one USD. However, this backing introduces centralization concerns, as the reserve USD is managed by traditional financial institutions, which requires placing trust in these entities.

Automated Market Makers (AMMs)

One example of decentralized services is Automated Market Makers (AMMs), such as Curve [20], Sushiswap[21], and Uniswap [22]. AMMs facilitate the automatic exchange of assets without a centralized authority, typically dealing with ERC20 tokens. These tokens utilize the *approve()* and *transfer()* functions to enable automatic trading by AMMs. In traditional settings, buyers and sellers place orders specifying prices and quantities for asset exchanges. A centralized exchange then matches these orders, with asset prices being determined by supply and demand.

In contrast, AMMs allow users to interact with a liquidity pool and a price function that algorithmically determines the asset's price. This setup enables users to exchange assets directly and atomically with an AMM without waiting for another user to match their desired trade proportions [XPCF23]. AMMs employ various price determination functions that define a price curve, ensuring the total value of the liquidity pool remains constant. For instance, Uniswap's constant product function $x * y = k$[23] means the product k must remain unchanged when trading assets x and y. Thus, a trade is only valid if the exact y amount is provided to keep k unchanged.
However, AMMs are subject to slippage, which is the difference between the expected price of a trade and the actual price executed. Slippage occurs when asset prices significantly fluctuate, leading to a difference between the price when a transaction is submitted to the MemPool and when it is included in a block. For instance, a large order can significantly alter the ratio of assets in the pool, leading to less favorable exchange rates for the trader. To mitigate slippage, traders can set a maximum acceptable slippage percentage when initiating a trade, ensuring that the trade will only execute if the price remains within the specified range.

[20] https://curve.fi/
[21] https://www.sushi.com/
[22] https://uniswap.org/
[23] https://docs.uniswap.org/contracts/v2/concepts/protocol-overview/how-uniswap-works

Lending protocols

Another significant DeFi application is lending protocols, such as Aave [24] and Compound [25]. Users can borrow assets by interacting with a lending Smart Contract and posting collateral, a security deposit of higher value than the borrowed asset [QZG+21]. Given the often volatile nature of cryptocurrency assets, the collateral's value may fall below the loan value. In such cases, the debt can be recovered through collateral Liquidations. These Liquidations can occur via auctions, where the collateral is available for bidding, or through fixed spread Liquidations, where the collateral is immediately available at a discount.

2.2.9 Conclusion

This chapter has established the technical foundation for this thesis by exploring key aspects of blockchain technology, with a particular focus on Ethereum. We began with an overview of fundamental blockchain concepts using Bitcoin as an example. This was followed by an in-depth examination of Ethereum, central to our study.

We detailed the mechanics of Ethereum addresses, transactions, and their propagation through the MemPool. We then explored the Ethereum Virtual Machine (EVM) and delved into Smart Contracts, illustrating their functionality with a specific example: the Flash Loan.

Given the significance of the Proof of Stake (PoS) mechanism to Maximal Extractable Value (MEV), we thoroughly explained its core components and operational principles. Another vital topic for MEV is decentralized finance (DeFi), which was introduced in this chapter. We introduced the ERC20 and ERC721 Token Standards and, finally, two significant DeFi Applications, Automated Market Makers and Lending Protocols.

[24] https://aave.com/
[25] https://compound.finance/

CHAPTER 3

MEV - Maximal Extractable Value

The ability of block proposers to manipulate transaction ordering and selectively include or exclude transactions has introduced a new attack vector in Decentralized Finance: Maximal Extractable Value (MEV). This chapter delves into the complexities of MEV, exploring its various forms and the broader implications for the blockchain ecosystem.

We introduce concrete examples of MEV attacks, demonstrating how they extract value at the expense of regular users. These attacks include Sandwich attacks, Arbitrage, Transaction Replays, and Clogging, illustrating how MEV can be leveraged to gain unfair advantages.

However, MEV's disadvantages extend beyond individual user's losses, affecting the overall health and security of the blockchain ecosystem. Therefore, this chapter will also explore these broader negative externalities, such as increased transaction costs, network congestion, and potential threats to consensus security. We will discuss how MEV can incentivize malicious behaviors among block producers, further undermining the stability of decentralized networks.

To address these challenges, the chapter concludes with an overview of various mitigation strategies. These strategies range from theoretical proposals to practical implementations currently in use. We will examine MEV auction platforms that aim to create transparent markets for block space, MEV-aware application designs that minimize vulnerabilities, and innovative approaches to achieving order fairness in transaction processing.

Overall, this chapter aims to introduce MEV and illuminate its pressing issues and ongoing efforts to develop effective solutions.

3.1 Characterization of MEV

In the following section, we will characterize various forms of MEV. Since the initial definition of MEV [DGK$^+$20], research has identified numerous MEV opportunities. We will present six types of MEV that target individual transactions: Frontrunning, Backrunning, Sandwich Attacks, Arbitrage, Transaction Replay, and Clogging. Following these simpler forms, we will provide a brief overview of a more complex form of MEV, Cross-Domain MEV.

Frontrunning

Frontrunning is a transaction ordering attack where the attacker aims to insert their transaction immediately before the victim's transaction within a block. One method to achieve this is by setting the gas price of their transaction just one unit higher than the victim's transaction. Since block proposers generally prioritize transactions by gas price, the attacker's transaction is expected to be included right before the victim's.

The result of a Frontrunning attack can be twofold: Tolerant and Destructive [QZG22]. In a destructive Frontrunning attack, the attacker alters the blockchain state to cause the victim's transaction to fail. In a tolerant Frontrunning attack, the victim's transaction remains valid but executes under less favorable conditions than initially expected.

A form of destructive Frontrunning is the displacement or transaction replay attack. For example, consider a lending contract where the loan value has dropped below the collateral's safety margin. Some lending contracts allow for immediate Liquidation of the collateral at a discount. If a victim identifies this opportunity and publishes a Liquidation transaction, an attacker monitoring the MemPool could see this transaction and publish an identical one with a higher gas fee. As a result, the attacker's transaction would be processed first, allowing them to benefit from the Liquidation and causing the victim's transaction to fail since the collateral would already be liquidated.

Backrunning

Contrary to a Frontrunning attack, a Backrunning attack involves the attacker aiming to include their transaction immediately after a targeted transaction to profit from the state changes introduced by the preceding transaction. For example, consider an Oracle update where new information is published significantly affecting an asset's price. By Backrunning the Oracle's update, the attacker can be the first to capitalize on this new information.

Sandwich Attack

A particularly lucrative form of attack is the Sandwich attack, which combines elements of both Frontrunning and Backrunning attacks. In an Automated Market Maker (AMM) exchange, users swap assets based on the AMM's current state, which determines the exchange rate. Due to the inherent delay introduced by the blockchain's consensus

mechanism — the time between publishing a transaction and its inclusion in a block — users typically set a slippage tolerance, which is the acceptable difference between the expected and actual prices. Transactions that exceed this slippage tolerance will automatically fail.
In a Sandwich attack, the attacker takes advantage of this slippage tolerance by publishing two transactions: before and after the victim's transaction.

One way an attacker could profit is the following:

1. The attacker's first transaction buys the same asset as the victim, driving up the asset's price.
2. The victim's transaction then executes at this higher price.
3. The attacker's second transaction sells the asset at the inflated price, thus profiting from the price difference.

This attack allows the attacker to profit by manipulating the asset's price to their advantage while the victim pays more than anticipated.

Arbitrage

Arbitrage trading involves making profits by exploiting price differences for the same asset across different exchanges. For example, if asset A costs 2.1 ETH on one decentralized exchange (DEX) but only 2 ETH on another, an arbitrage trader can buy the asset for 2 ETH and immediately sell it for 2.1 ETH, realizing a profit of 0.1 ETH.

There are various techniques for arbitrage trading. A common method is Backrunning a transaction, but another approach is blocked state arbitrage, where a trader monitors blockchain states and then Frontruns all other transactions in the next block if an arbitrage opportunity arises [QZG22]. Due to the complexity of the DeFi ecosystem, arbitrage opportunities can involve multiple exchanges or tokens [1] [PJL$^+$23], and even span multiple blockchains [OSS$^+$21]. Park et al. [PJL$^+$23] have also researched arbitrage opportunities that leverage Smart Contract-specific structures, such as burn and mint mechanisms, or the coordination of multiple MEV bots to maximize profits.

Transaction Replay

In a transaction replay attack, an attacker monitors the MemPool for public information in unconfirmed transactions. The attacker then acts on this information before the initial transaction is confirmed. For instance, they might copy and modify a transaction to redirect potential profits to a wallet they control. A notable example of this type of attack was described by Robinson and Konstantopoulos [RK20]. In this case, someone

[1] https://www.odos.xyz/arbitrage

accidentally sent tokens to a Smart Contract that allowed anyone to retrieve those tokens by calling a specific function.

Robinson and Konstantopoulos attempted to call this function to rescue the tokens. However, by doing so, they inadvertently exposed the vulnerability. An attacker observing the MemPool noticed this vulnerability and quickly published a transaction to exploit it, resulting in the loss of the tokens to the attacker.

Clogging

A clogging attack is a censorship attack where an adversary spams the blockchain with transactions to prevent others from issuing their transactions. Imagine a Smart Contract-based game or lottery in which an attacker gains an advantage by flooding the blockchain with their transactions, blocking others from interacting with the Smart Contract.

An example of a clogging attack occurred during the first round of the game Fomo3D [EMC20]. Fomo3D is a game where the winner is the last person to buy a ticket before the timer runs out. Each ticket purchase extends the timer by 30 seconds. In the first round, many players used automated scripts to buy tickets just before the timer ended, potentially prolonging the game indefinitely. However, on August 22, 2018, a player won the first iteration of this game and was rewarded 10,469 Ether. The winner employed a clogging attack by purchasing a ticket and then submitting numerous high gas price transactions. Miners prioritized these transactions due to their higher fees, including them in the subsequent blocks. Given Ethereum blocks have a total gas limit [2], the other players' purchases were excluded from these blocks. This allowed the attacker to secure the last ticket purchase and ultimately win the game.

3.2 Multiple Dimensions of MEV

The complex nature of blockchains and DeFi has created numerous opportunities for adversaries to profit by reordering transactions. While the previous examples focused on relatively simple attacks targeting single transactions, the following sections will briefly discuss two more sophisticated MEV opportunities: Cross-Domain MEV and Multi-block MEV.

Cross-Domain MEV

To the best of our knowledge, Obadia et al. [OSS+21] were the first to formalize the concept of cross-domain MEV. Cross-domain MEV refers to the value that can be extracted by influencing transaction orders across multiple blockchains. DeFi applications like Automated Market Makers are not confined to a single blockchain like Ethereum. For instance, multiple instances of Sushiswap operate on different blockchains.

[2]https://ethereum.org/en/developers/docs/gas/#block-size

Different blockchains use various consensus mechanisms and protocols, making cross-chain communication challenging. Bridges are protocols that facilitate communication between blockchains and allow users to move assets across them.

As AMMs exist on different blockchains, they likely have different liquidity pool sizes and prices for the same assets. Any imbalance in these factors across blockchains creates cross-domain arbitrage opportunities. Unlike simpler MEV examples, cross-domain MEV is complicated because cross-domain trades are not atomic.

The importance of atomic cross-domain transactions is particularly evident when considering arbitrage opportunities, as described in section 3.1. In these scenarios, a user or bot relies on completing two transactions: a buy order (e.g., for 2 ETH) and an immediate sell order (e.g., for 2.1 ETH). If the two unbalanced decentralized exchanges are on different blockchains, the lack of atomicity increases the risk that only one or neither of the transactions will be finalized. This risk is further compounded if multiple bots compete for the same arbitrage opportunity.

Multi Block MEV

In Ethereum, validators and committee members are selected using the RANDAO mechanism, which pseudo-randomly and unpredictably selects these roles. However, at the end of epoch $N-1$, the proposers for epochs N and $N+1$ are known [JvWR23]. It is possible, and not uncommon, for up to five or even seven consecutive slots to be assigned to a single pool. Additionally, mechanisms such as MEV Boost [3] allow users to pay fees to validators to include proposed blocks (see section 3.0.5). This setup allows MEV searchers to influence multiple blocks to exploit MEV opportunities.

An example of a Multi-block MEV extraction strategy involves a builder censoring all sell transactions to an AMM, thereby increasing the asset's price. After this period, the searcher can Frontrun all other transactions by selling their assets at an inflated price. The next naive proposer would then naturally include all the previously censored sell transactions, resulting in a balanced price.

3.3 Negative externalities of MEV

Most previously discussed examples highlight the direct negative impacts of MEV on individual victims. For instance, a user who buys assets through an AMM and is subjected to a Sandwich attack ends up paying a much higher price than anticipated. However, MEV also generates broader negative externalities that can affect all blockchain users and compromise the security of the blockchain consensus.

Daian et al. [DGK+20] illustrated the issue of Priority Gas Auctions (PGAs). Many MEV opportunities represent pure revenue possibilities, often because executing multiple

[3]https://github.com/flashbots/mev-boost

actions atomically within a Smart Contract is possible. These pure profit opportunities lead to competition among MEV bots.

To secure these opportunities, MEV bots frequently raise the gas prices of their transactions to ensure inclusion in the finalized blockchain. This behavior leads to network congestion, reduced transaction throughput, and increased gas prices for all users. Moreover, MEV can incentivize attacks on consensus security. Block producers capture many MEV profits [MKV23], which might tempt them to alter blockchain history to capitalize on past MEV opportunities.

One such attack is the undercutting attack, where a miner forks a high-fee block and withholds fees to attract other miners to build on the new fork [CKWN16]. Another potential attack is the time bandit attack, where an attacker identifies lucrative MEV opportunities in a past block. If the MEV opportunity is substantial, the attacker forks the block and uses rented hash power to rewrite the blockchain history up to the current height, ensuring the new fork is accepted by all clients [DGK+20].
Qin et al. [QZG22] found that when MEV is four times higher than the block reward, a miner with 10 percent mining power might prefer forking the blockchain over honest mining. Between December 1, 2018, and August 5, 2021, they identified at least 2,407 blocks with MEV opportunities that met this condition. In another study, Piet et al. [PFW22] discovered four blocks in 12 days that would have made time bandit attacks profitable.
To shield themselves from MEV attacks, many users might resort to private communication with mining or staking pools, leading to increasingly centralized and permissioned systems. This shift could raise entry barriers for users without access to such systems. Furthermore, private communication between entire trading firms and staking pools could increase centralization within the ecosystem [HG22].

3.4 Mitigation

As discussed in section 3.3, MEV poses a significant problem that degrades the user experience on blockchain networks and threatens the security of the underlying consensus mechanism.

Since the publication of the Flashboys 2.0 paper [DGK+20], numerous approaches to address and mitigate the effects of MEV have been proposed. Some of these proposals have been implemented and are actively used by many users, while others remain theoretical research outputs.

This section introduces three proposed Solutions to mitigate the negative externalities of MEV: Auction Platforms, Order Fairness, and MEV-aware application Design.

3.4.1 MEV Auction Platforms

MEV auction platforms do not primarily aim to prevent MEV exploitation; they seek to make MEV activities transparent, distribute MEV profits more equitably, and mitigate some of MEV's negative externalities. By creating a market for block space, these platforms help to avoid the vertical integration of block producers, MEV searchers, decentralized exchanges (DEXs), and other entities [HG22].

MEV auction platforms typically allow users to bid on block space. Proposers sell block space to MEV exploiters/searchers or regular users. These platforms generally promise privacy for submitted bundles/transactions and ensure atomicity (either the entire bundle is included or none is) [YZH+22]. MEV searchers use these platforms to secure their MEV opportunities without revealing their transactions before confirmation. Regular users might use them to maintain privacy, especially from MEV searchers.

Creating a market for block builders, Proposer-builder separation (PBS), is planned to be integrated into the Ethereum protocol itself [VNSA+21]. In the meantime, several projects have implemented forms of PBS, the most prominent being Flashbots'[4] MEV-Boost [5].

In MEV-Boost, block builders construct blocks based on the public MemPool and potentially their MEV-exploiting transactions and then submit them to one or more relays. These blocks typically include a fee paid to the block proposer if they choose the block. The relays select the block with the highest fee and forward it to connected block proposers, who publish these blocks. To prevent proposers from ignoring the proposed block and using the contained information themselves, the relay sends only the block's header information to the proposer. Only after the proposer has signed this so-called blinded block [6] does the relay reveal the full block body, including its transactions. Thus, the relays remain trusted entities upon which both block builders and proposers rely.

While these auctions create a competitive market for builders, potentially reducing negative externalities such as centralization efforts and network congestion, the MEV profits primarily benefit block builders and proposers, not the users who created the MEV opportunities.

In the following, we explore two solutions designed to share MEV profits with users:

- Transaction Auctions by M. Köppelmann [Kö23]
  This approach focuses on transaction-level auctions instead of auctioning entire blocks. In these auctions, users sell the slot immediately following their transaction. MEV searchers bid on these Backrunning opportunities and must pay the users based on the auction outcome.

- MEV Share by Flashbot [BAH+23]
  MEV Share follows a similar strategy. Users have control over which transaction information to reveal, such as omitting the contract address, calldata, or logs

[4]https://www.flashbots.net/
[5]https://github.com/flashbots/mev-boost
[6]https://ethereum.github.io/builder-specs/#/Builder/submitBlindedBlock

> included in a transaction [7]. A middleware called Matchmaker shares selective transaction data, respecting the users' privacy preferences. MEV searchers then submit bundles of transactions, similar to MEV-Boost, with optional indications of where the private transactions might be included. The Matchmaker attempts to match these bundles with the users' private transactions by simulating various combinations to maximize MEV profits. These matched bundles are sent to builders with the condition that users receive a portion of the generated MEV. Initially, this condition is not enforced permissionlessly, so the Matchmaker only interacts with trusted builders. This process could be enhanced using trusted hardware or cryptoeconomic [8] solutions such as security bonds posted by builders.

MEV auction platforms are among the most widely adopted mitigation strategies, particularly the open-source MEV-Boost [9] platform by Flashbots. According to a dashboard provided by Anton Wahrstätter, over 90% of blocks are published using MEV-Boost [10]. While auction platforms do not eliminate MEV or all of its negative externalities, they at least make MEV activities transparent and help mitigate some centralization forces.

3.4.2 Order Fairness

The next class of mitigation strategies aims to address the problem of MEV by achieving fair ordering. The primary objective is to prevent single malicious actors from manipulating transaction ordering. There are two main approaches to achieving this. One approach to achieve this is time-based order fairness. The goal here is that if a transaction T_1 is received by a majority of honest nodes before transaction T_2, then T_1 should be placed before T_2 in the confirmed block [KZGJ20]. This ensures transactions are ordered based on the time they are received by the network, reducing the potential for manipulation.

Another approach is content-agnostic ordering. This method prevents order manipulation based on transaction content by ensuring that those responsible for ordering transactions do not know the details of the transactions.

Time-Based Order Fairness

As described before, strict time-based order fairness is a FIFO mechanism. It is, however, impossible to achieve this, as shown by Kelkar et al. [KZGJ20]. The following is their example, which shows that it is impossible in an asynchronous network.

Three nodes, A, B, and C receive transactions $[x, y, z]$ in the following order: A: $[x, y, z]$, B: $[y, z, x]$, and C: $[z, x, y]$. In this example, most nodes received x before y, but most also received y before z and z before x. This example called the Condorcet paradox

[7] https://docs.flashbots.net/flashbots-auction/searchers/advanced/rpc-endpoint#eth_sendprivatetransaction

[8] https://policyreview.info/pdf/policyreview-2021-2-1553.pdf

[9] https://github.com/flashbots/mev-boost/

[10] 14-day share of 91.6% https://mevboost.pics/, accessed on 2023-12-01

[DC$^+$85], makes it obvious, that it is not possible to define a protocol in which the majority of nodes, two of three nodes, agree on the same ordering.
Therefore, mitigation strategies have to weaken the assumption of order fairness.
Protocols proposed are for example, the Fair Sequencing Service (FSS) by Chainlink[11], the espresso sequencer[12], or the Aequitas protocols by Kelkar et al.[KZGJ20]. While these and other protocols achieve increasing levels of fairness, they are prone to attacks by hackers with better network connections [HW22]. Attacks can always listen to the network and Frontrun transactions with lower latency than the honest nodes. Additionally, Backrunning is even harder to prevent, although its effects are generally less severe than the effects of Frontrunning [HW22].

Content-Agnostic Ordering

With content-agnostic ordering, cryptographic techniques ensure that the nodes responsible for ordering transactions do not know their content. Typically, users commit to their transactions cryptographically, revealing only certain data fields, such as fees, to the ordering nodes [YZH$^+$22]. The ordering nodes then arrange the transactions based on their receipt order. In a subsequent reveal phase, the encrypted content is decrypted and executed according to the predetermined order.

Several techniques support this commit-and-reveal scheme, including threshold encryption, timelock encryption, and off-chain methods [HW22].

Threshold encryption can be employed to prevent users from selectively revealing only beneficial transactions. In this scheme, a committee of n members generates a public key that users use to encrypt their transactions. These transactions can then be decrypted by l out of n committee members. Examples of threshold encryption implementations include Shutter Network [13], Osmosis [14], and Sikka [15].

Another approach is time-locked encryption, which ensures that the decryption of commitments occurs after a probabilistically set time [RSW96]. Protocols such as TeX [KGF19], a trustless exchange, and Multi-Party Timed Commitments [DE20], are based on time-locked encryption.

The downside of commit-and-reveal schemes is that they often operate on the blockchain, consuming more space than simple transactions and increasing transaction costs. Additionally, there is a significant delay between the commit and the reveal phases and the final confirmation of the transactions. This delay might be impractical for users trading volatile assets [HW22].

[11]https://chain.link/
[12]https://hackmd.io/@EspressoSystems/EspressoSequencer
[13]https://blog.shutter.network/
[14]https://osmosis.zone/
[15]https://sikka.tech/projects/

3.4.3 MEV-Aware Application Design

This section explores design choices that decentralized exchanges and Smart Contracts can implement to mitigate MEV issues. Generally, these design choices are tailored to the specific types of MEV that a Smart Contract expects to encounter.

One straightforward MEV mitigation strategy is setting a limit on gas prices within a Smart Contract [EMC20]. This strategy caps the maximum gas price for transactions interacting with the contract, aiming to prevent Frontrunning attacks that rely on Priority Gas Auctions (PGAs). Users can set their gas prices at or near the limit, thus preventing other transactions from bypassing theirs by offering a higher gas price.

Another method to minimize MEV attack surfaces is adjusting slippage parameters to smaller values. However, this involves a trade-off: lower slippage reduces MEV exposure but increases the risk of transaction failure, while higher slippage increases MEV risk [YZH+22].

To counter the threat of Backrunning, the A2MM market maker automatically collects arbitrage opportunities within the transaction interacting with the customer [ZQG21]. Another approach is to conduct part of the matchmaking process offline, as Hashflow [16] does. While this strategy mitigates MEV risk, it moves the reordering attack surface off-protocol.

Executing auctions in batches is another MEV-aware strategy. By aggregating transactions into batches at discrete time intervals, all transactions within those batches are executed simultaneously [YZH+22]. CowSwap [17] is an example of a market maker employing batch transactions.

One idea for preventing Sandwich attacks in the context of decentralized exchanges is to only allow trades against a fixed state. This concept is inspired by Bitcoin's unspent transaction output (UTXO) model. In Bitcoin, users do not have account balances but spend the outputs from previous transactions consumed in new transactions. The eUTXO model proposes storing liquidity pools in UTXOs [Lan21]. When transacting, a user must specify a particular eUTXO liquidity pool, ensuring the transaction is executed in a specific state and cannot be executed in another. This approach is akin to setting the slippage parameter to zero. However, it significantly limits throughput, allowing only one transaction per block.

While the previous examples offered technical approaches to mitigating MEV through application design, the example shown in Figure 3.1 illustrates an MEV-aware graphical user interface (GUI) design. Before executing a trade within this GUI, users are alerted that Sandwich attacks commonly target this particular trading pair. The interface warns users and recommends reducing the slippage tolerance to mitigate the risk of such attacks.

[16] https://www.hashflow.com/

[17] https://swap.cow.fi

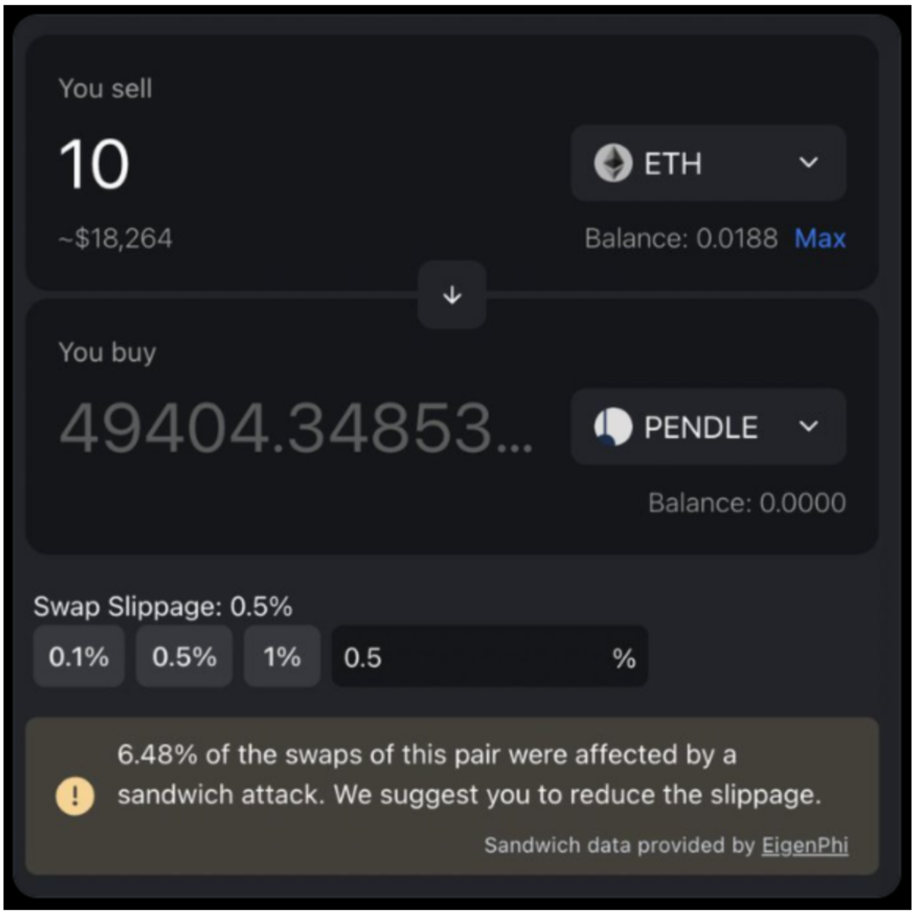

You sell
10
~$18,264
ETH
Balance: 0.0188 Max
You buy
49404.34853...
PENDLE
Balance: 0.0000
Swap Slippage: 0.5%
0.1% 0.5% 1% 0.5 %
6.48% of the swaps of this pair were affected by a sandwich attack. We suggest you to reduce the slippage.
Sandwich data provided by EigenPhi

Figure 3.1: An example of MEV-Aware application design. [vro23]

3.5 Conclusion

This chapter has provided an in-depth exploration of Maximal Extractable Value (MEV) and its significant implications for blockchain and Decentralized Finance (DeFi) ecosystems.

We identified two categories of MEV: Single-transaction MEV and Multi-Dimensional MEV. Single-transaction MEV includes Frontrunning, Backrunning, Sandwich attacks, arbitrage, transaction replay, and clogging. Multi-dimensional MEV encompasses Cross-Domain MEV and Multi-block MEV.

While there is an obvious disadvantage to users targeted by MEV attacks, broader negative externalities also exist. These include network congestion, increased gas prices, and threats to consensus security through reordering attacks.

Various mitigation strategies have been proposed and implemented to address these challenges. MEV auction platforms aim to make MEV activities more transparent and distribute MEV profits more equitably. Order fairness protocols seek to ensure that transactions are processed fairly and transparently. MEV-aware application design minimizes the attack surface for Smart Contracts and decentralized exchanges.

CHAPTER 4

A systematic literature review of state-of-the-art MEV detection

To confidently understand Multi-block MEV, it is essential to understand MEV detection in a Single-block setting. This chapter aims to investigate the current state-of-the-art of MEV detection and quantification. It focuses on answering the following research question:

> *RQ_2: What are the state-of-the-art methods for detecting and quantifying Single-block Maximal Extractable Value in Ethereum, and what are the limitations of these methods?*

To do this, we will review current literature and MEV detection tools to understand the techniques used to quantify MEV.

The review is conducted through a systematic literature review guided by the PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) guidelines [PMB+21a, PMB+21b].

Our literature search encompassed Google Scholar, arXiv, TU Wien Library, ACM Digital Library, IEEE Xplore, and Springer Link. Inclusion criteria included studies on MEV detection, quantification, and qualification within the Ethereum blockchain. We excluded studies related to MEV in other blockchains and papers published in languages other than English and German.

From an initial pool of 1004 papers, 16 were deemed eligible. These selected studies revealed two predominant MEV detection approaches: rule-based and machine-learning-based methods. Our analysis thoroughly examines these approaches, highlighting their methodologies, strengths, and limitations.

This chapter will first present the steps of our systematic literature review before summarizing and classifying the select papers.

4.1 Methodology

To answer the research question, we will conduct a systematic review of the current state-of-the-art. This review will be guided by the PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) statement [PMB+21a, PMB+21b], which provides a framework for transparent and comprehensive reporting of systematic reviews. By adhering to the PRISMA guidelines, we aim to enhance the reproducibility and reliability of our review, ensuring that it is detailed enough to assess its trustworthiness and the applicability of its findings.

The PRISMA statement provides specific guidelines and offers resources such as a checklist and a tool [1] for generating a flow diagram that visualizes the different phases of the review process. These resources are available on the PRISMA website [2]. We will utilize the checklist and the flow diagram tool [3] and document their use in this thesis to provide a clear and structured overview of our review process.
The checklist is documented in the Appendix, in Table 1.

Although PRISMA's primary focus is to provide guidelines for systematic reviews in the healthcare sector, its principles are widely applicable to systematic reviews in other fields, including blockchain and cryptocurrency research [PMB+21b]. Following these guidelines ensures that our review of Single-Block MEV detection methods in Ethereum is conducted rigorously and transparently, providing valuable insights into the current methodologies and their limitations.

4.1.1 Inclusion and Exclusion criteria

The following inclusion and exclusion criteria were used when searching for literature handling MEV detection.

Inclusion criteria

The work focuses on the following topics:

- MEV detection in Ethereum
- MEV quantification in Ethereum
- Analysis of MEV searcher strategies in Ethereum
- The work is a Scientific Paper, open-source Tool or Blog-Entry

[1]https://www.eshackathon.org/software/PRISMA2020.html
[2]https://www.prisma-statement.org/ Accessed on 2024-07-17
[3]https://www.eshackathon.org/software/PRISMA2020.html

- It focuses on MEV extraction on layer 1 Ethereum
- The main focus of this work is to detect and quantify MEV, or quantifying MEV is an important part of the work
- The work quantifies one of the following forms of MEV: Priority Gas Auctions, Sandwich attacks, Frontrunning, Backrunning, Clogging, Atomic Arbitrage, Replay
- The work quantifies MEV transactions using on-chain data
- The work reports the concrete strategy used to quantify MEV

Exclusion criteria

- The work is not written in English and German
- The work is not publically available for a student using the TU Wien network
- The work focuses on another blockchain besides Ethereum
- The work does not describe how MEV is calculated but only presents data
- The work title does not indicate that the work handles MEV detection.
- The work introduces a new protocol or system and does not focus on measuring MEV
- The work is a whitepaper, introducing a new system

4.1.2 Sources

For this research, online resources were exclusively used. The following resources were used:

- Google Scholar [4], last search on 2024-07-18
- Arxiv [5], last search on 2024-07-18
- TU Wien Library [6], last search on 2024-07-18
- ACM Digital Library [7], last search on 2024-07-18
- IEEE Xplore [8], last search on 2024-07-18
- Springer Link [9], last search on 2024-07-18

[4]https://scholar.google.com/
[5]https://arxiv.org/
[6]https://catalogplus.tuwien.at/primo-explore/search
[7]https://dl.acm.org/
[8]https://ieeexplore.ieee.org/Xplore/home.jsp
[9]https://link.springer.com/

4.1.3 Search strategy

The PRISMA guidelines recommend providing a complete search strategy for each source. A previous version of these guidelines [MLTA09] suggested that the strategy be presented for only one source. However, we will present the search strategy for all sources to enhance the reader's ability to reproduce and understand it. This aligns with the revised and updated PRISMA guidelines [PMB+21a]. Table 4.1 lists the exact search strings for each source.

Source	Search String
Google Scholar	("Maximal Extractable Value" AND "Ethereum") AND ("quantification" OR "mea- surement" OR "analysis" OR "evaluation" OR "assessment")
Arxiv	order: announced_date_first; size: 50; date_range: from 2015-01-01 ; classification: Computer Science (cs); include_cross_list: True; terms: AND abstract="Ethereum" OR "Smart Contract"; AND all="quantification" OR "measurement" OR "analysis" OR "evaluation" OR "assessment"; AND abstract="Maximal Extractable Value" OR "MEV"
TU Wien Library	query=title,contains,"Maximal Extractable Value" OR "Miner Extractable Value",AND&query=title,contains,"quantification" OR "measurement" OR "analysis" OR "evaluation" OR assessment",AND&pfilter=dr_s,exact,20150101,AND&pfilter=dr_e, exact,99991231
ACM Digital Library	[[Abstract: maximal extractable value] OR [Abstract: miner extractable value] OR [Abstract: ethereum]] AND [[Title: quantification] OR [Title: measurement] OR [Title: analysis] OR [Title: evaluation] OR [Title: assessment]] AND [E-Publication Date: (01/01/2015 TO *)]
IEEE Xplore	("Abstract":MEV OR "Abstract":Maximal Extractable Value OR "Abstract":Miner Extractable Value) AND ("Abstract":Ethereum) AND ("Full Text & Metadata":quantification OR "Full Text & Metadata":measurement OR "Full Text & Metadata":evaluation OR "Full Text & Metadata":assessment)
Springer Link	"Ethereum" AND ("Maximal Extractable Value" OR "MEV") AND ("quantification" OR "measurement" OR "evaluation" OR "assessment")

Table 4.1: The search strings used for each source in this systematic review.

4.1.4 Selection Process

According to PRISMA Item 8, the selection process should be clearly defined, "including how many reviewers screened each record and each report retrieved, whether they worked independently, and, if applicable, details of automation tools used in the process" [PMB+21b]. Only one person was available to screen all the results for this study.

First, all data was imported into the tool JabRef [10]. JabRef has a built-in function for entering a search query, automatically querying search engines, and importing the results. This, however, only worked in the case of the ACM Digital Library. For all other cases, the JabRef Chrome Browser Extension was used to import entries from all study sites manually. A total of 1004 results were imported into JabRef. Using JabRef's duplication removal tool, duplicate entries were removed, resulting in 787 entries.
The initial selection was based on reviewing the titles of all works. Entries were removed if they did not meet the inclusion criteria or if they met any exclusion criteria. After this first screening, 118 entries remained.

All abstracts were read in the second round of the selection process to determine whether the work met the inclusion/exclusion criteria. This second round resulted in 27 remaining entries. Among these, 4 duplicates not previously detected by JabRef were found, resulting in 23 entries.
In the third and last round, all 23 entries were read to determine if they were relevant to this study. In this round, 16 entries were deemed relevant for this systematic study. Table 4.2 lists all selected works.

Table 4.2: All works retrieved during the systematic selection process

	Title	**Authors**	**Year**
[Wun23]	Exploring Maximal Extractable Value in the Ethereum Ecosystem	Wunderlich, Sebastian	2023
[TC+21]	Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain.	Torres et. al	2021
[PFW22]	Extracting Godl [sic] from the Salt Mines: Ethereum Miners Extracting Value	Piet et. al	2022
[CHHW24]	Remeasuring the Arbitrage and Sandwich Attacks of Maximal Extractable Value in Ethereum	Chi et. al	2024
[WTNRS22]	A flash(bot) in the pan: measuring maximal extractable value in private pools	Weintraub et. al	2022
[ZNW21]	Analyzing and preventing sandwich attacks in ethereum	Züst et. al	2021

Continued on next page

[10] https://www.jabref.org/

	Title	**Authors**	**Year**
[YHL+]	Mecon: A Gnn-Based Graph Classification Framework for Mev Activity Detection	Yao et. al	2022
[DGK+20]	Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability	Daian et. al	2020
[QZG22]	Quantifying blockchain extractable value: How dark is the forest?	Qin et. al	2022
[Han22]	Arbitrage in crypto markets: An analysis of primary ethereum blockchain data	Hansson, Magnus	2022
[LZWD24]	A Geth-based real-time detection system for sandwich attacks in Ethereum	Li et. al	2024
[MKV23]	A large scale study of the ethereum arbitrage ecosystem	McLaughlin et. al	2023
[LLPL24]	GasTrace: Detecting Sandwich Attack Malicious Accounts in Ethereum	Liu et. al	2024
[PJL+23]	Unraveling the MEV Enigma: ABI-Free Detection Model using Graph Neural Networks	Park et. al	2023
[Eig24a]	EigenPhi (Tool)	Cheng et. al	n/a
[SHM21]	MEV-Inspect (Tool)	Miller et. al	n/a

4.1.5 Data Extraction Process

Per the PRISMA Statement [PMB+21a], we outline the Data Extraction Process. One person conducted the Data Extraction Process. All 16 entries were reviewed, and the following questions were systematically answered for each work to extract the necessary information:

- Which exact forms of MEV does this work quantify?
- In what period (time, blocks) does this work quantify MEV?
- Does the work use solely On-Chain data? If not, what external data does this work use?
- What is the exact strategy to detect MEV transactions?

In addition, we extracted the following metadata for each work:

- Full Title

- Author Name(s)
- Type (Tool, Conference Paper, Thesis,...)
- Date of Publication

4.1.6 Risk of bias assessment and Synthesis methods

The following questions were asked to assess the risk of bias, as required by the PRISMA statement. These questions are loosely based on the Joanna Briggs Institute (JBI) Critical Appraisal Tools [AFG+15]. As with the systematic review, one person was used to analyze these questions. The goal was to have an additional assessment of the quality of the work. The result can be seen in Table 4.3.

- **Q1** Did the authors justify the selection of the study timeframe and ensure that it is appropriate for capturing relevant data on MEV quantification?
- **Q2** Did the authors report their funding sources, and/or did they provide an assessment of how the funding might influence the results?
- **Q3** Did the authors report any potential conflicts of interest and discuss how these might impact the study's findings?
- **Q4** Did the authors provide a clear and detailed description of the study design and methodology for MEV quantification?
- **Q5** Were the data collection methods and analysis techniques appropriate and adequately described to ensure applicability?
- **Q6** Did the authors address any potential biases or limitations in the data collection process?
- **Q7** Did the authors provide access to their data or analysis code, if applicable, to allow for verification of results?

As the PRISMA guidelines primarily focus on understanding medical, mostly empirical studies, they also focus on synthesizing data and quantitative measures. In this study, however, we want to focus on understanding the techniques used to extract and quantify MEV. We explicitly do not want to cumulate or synthesize the measured MEV transactions. Therefore, we will apply a narrative synthesis [PRS+06].
To this goal, all selected papers will be read and categorized based on their applied MEV detection category. Based on that, we will compare all techniques based on complexity, accuracy, and maintainability and highlight strengths and weaknesses.

4.1.7 PRISMA Flow diagram

We present the PRISMA Flow diagram for this review in Figure 4.1. It was created with the PRISMA Flow diagram tool [HPPM22].

Paper	Qualtiy and Bias Criteria						
	Q1	Q2	Q3	Q4	Q5	Q6	Q7
[Wun23]	✓	✗	✗	✓	✓	✗	✓
[PFW22]	✗	✗	✗	✓	✓	✗	✗
[CHHW24]	✗	✗	✗	✓	✓	✗	✓
[WTNRS22]	✗	✓	✗	✓	✓	✓	✓
[ZNW21]	✗	✗	✗	✓	✓	✗	✓
[YHL$^+$]	✓	✗	✗	✓	✓	✓	✗
[DGK$^+$20]	✗	✓	✗	✓	✓	✓	✓
[QZG22]	✗	✗	✗	✓	✓	✓	✗
[Han22]	✗	✗	✗	✓	✓	✗	✗
[LZWD24]	✗	✓	✓	✓	✗	✗	✓
[MKV23]	✓	✓	✗	✓	✓	✓	✓
[LLPL24]	✗	✓	✗	✗	✗	✗	✓
[PJL$^+$23]	✓	✗	✗	✓	✓	✓	✓

Table 4.3: Result of the Bias and Quality assessment (Questions are from subsection 4.1.6)

4.1.8 Excluded studies

Following the PRISMA guidelines Item # 16b, we will list items that "might appear to meet the inclusion criteria, but which were excluded" [PMB$^+$21b]. Due to the large number of exclusions, we will provide a representative selection illustrating various exclusion reasons.
Sjursen et al. [SMC23] wrote a conference paper on quantifying Cross-Domain MEV Value using Uniswap data from different domains to detect cross-domain arbitrage. Although the work focuses on MEV transactions, it was excluded because it dealt with cross-domain MEV, which is not the focus of this study.
Other examples of studies excluded due to their focus on different domains of MEV include works by Torres et al. [TMW$^+$24], Yan et al. [YLK$^+$24], and Ilisei et al. [Ili24].
Chaurasia et al. [CDG$^+$24] provide an overview of the evolution of the MEV ecosystem, from Priority Gas Auctions as described by Daian et al. [DGK$^+$20] to the present SUAVE [11] by Flashbots. However, this work does not introduce a novel method for quantifying MEV.
Öz et al. [ÖKV$^+$23] offer an insight into the role of time in the context of MEV. This study was excluded due to its focus on timing games, which is not the focus of this study.
Ji et al.. [JG24] Several papers focus on the legal or regulatory aspect of MEV. Our study focuses exclusively on MEV quantifying, so we have excluded such papers.
Öz et al. The article "A Study of MEV Extraction Techniques on a First-Come-First-Served Blockchain" by Öz et al. [ÖRG$^+$24] focuses on MEV extraction techniques in the blockchain Algorand. Another example would be the work of Carrillo et al. [CH23]

[11]https://writings.flashbots.net/mevm-suave-centauri-and-beyond Accessed on 2024-07-23

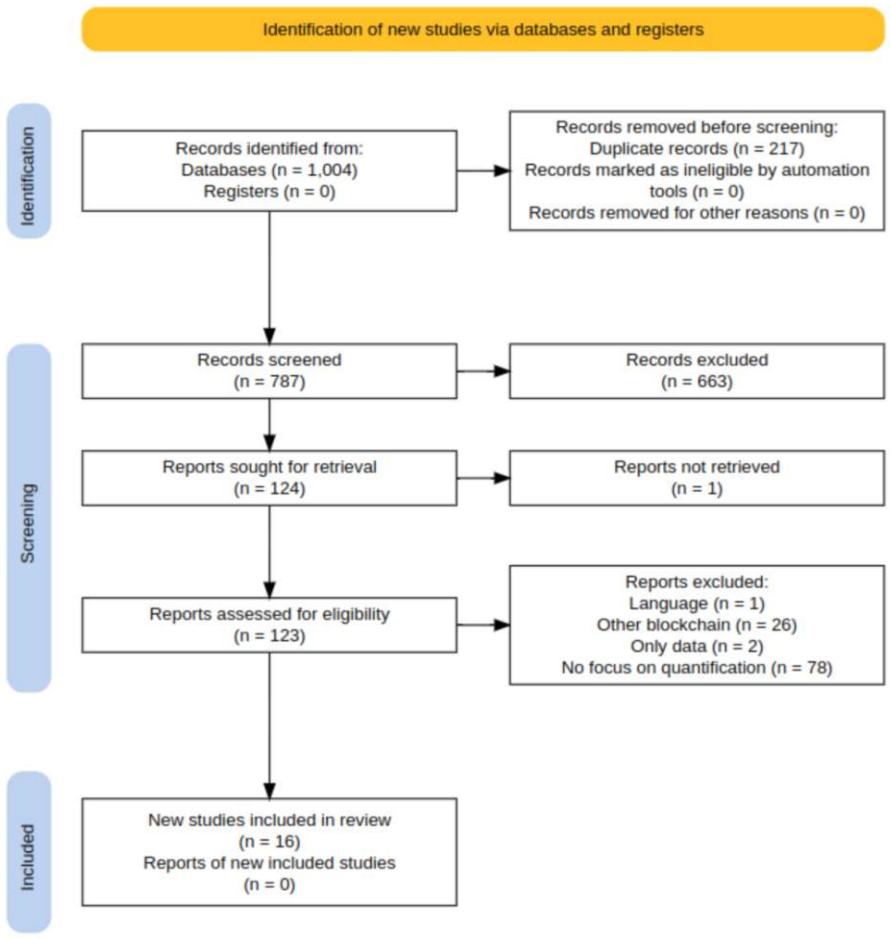

Identification of new studies via databases and registers
Identification
Screening
Included
Records identified from:
Databases (n = 1,004)
Registers (n = 0)
Records removed before screening:
Duplicate records (n = 217)
Records marked as ineligible by automation tools (n = 0)
Records removed for other reasons (n = 0)
Records screened
(n = 787)
Records excluded
(n = 663)
Reports sought for retrieval
(n = 124)
Reports not retrieved
(n = 1)
Reports assessed for eligibility
(n = 123)
Reports excluded:
Language (n = 1)
Other blockchain (n = 26)
Only data (n = 2)
No focus on quantification (n = 78)
New studies included in review
(n = 16)
Reports of new included studies
(n = 0)

Figure 4.1: The PRISMA Flow diagram, generated with [HPPM22]

focusing on MEV in Terra Classic. This study focuses exclusively on Ethereum, which is why they were excluded.

Yan et al. Yan et al. use data science methods to quantify Proof-of-Stake rewards. While they mention that MEV is an income source for Stakers in Proof of Stake Ethereum, they do not focus on quantifying it but on other rewards, such as attestation, proposer, and sync committee rewards.

Mao et al. [MZVL24] Is an example of several excluded papers due to their missing focus on MEV quantification. Although they offer some MEV quantification, they focus on introducing an MEV mitigation mechanism rather than presenting a novel method.

Heimbach et al. This work [HPS24] focuses on identifying Non-Atomic Arbitrage in DeFi. This work was excluded as this study focuses on atomic arbitrage detection, meaning arbitrages within a single transaction.

Zhou et al. focuses on identifying arbitrage *opportunities* [ZQC$^+$21], not the analysis of such transactions once they were confirmed.

4.2 Anylysis of the selected works

The following section will introduce the methods described in the selected works.

Daian et al.

One of the first papers to shed light on the problem of MEV was the work of Daian et al. [DGK$^+$20], called *Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges*.
This paper focused on a specific form of MEV, so-called pure revenue opportunities. Pure revenue opportunities are possible in Ethereum since Smart Contracts can invoke multiple other Smart Contracts in a single transaction. If any of those invocations fails, the whole transaction is reverted, guaranteeing an atomic execution of all invocations. With the help of this aspect, bots can now create transactions that are guaranteed to profit or fail. However, since these risk-free guaranteed profits exist, competition among MEV bots emerges. These bots increase their bribe bids in the form of gas prices to motivate proposers to have their transactions included on-chain. An economically rational block proposer is expected to prioritize transactions with a higher bribe. Daian et al. named this process Priority Gas auctions (PGAs).

Since only the winner of a PGA is included in the blockchain, Daian et al. could not simply investigate on-chain data to quantify PGAs. Instead, they deployed six geographically distributed nodes that recorded transactions relayed to them. These nodes ran a modified version of the GETH client [12], since in a PGA, transactions would often be replaced, and the standard GETH client would not relay replaced transactions. Storing every transaction relayed to these nodes would be infeasible. Therefore, they focused on a list of pre-defined suspected MEV bots. In addition, this list was updated every time a transaction was seen with gas prices, which were considerably higher than the current market level.
To detect all transactions in the auction, they used heuristics to identify transactions within a time frame around an observed winning PGA transaction. They also aggregated data to calculate the latency trends of the observed bots.
In addition to quantifying PGAs, they also provided a conservative estimate of pure revenue opportunities.

Their data shows that not every pure revenue opportunity results in a PGA, meaning that bots can profit uncontested from these opportunities. To provide a conservative estimation of pure revenue opportunities, they inspected transaction logs to detect transactions that contain more than two trades executed by a Smart Contract. A pre-defined list of known DEXes was used, meaning that transactions from unsupported DEXs were ignored.

[12]https://geth.ethereum.org/

Qin et al.

The work of Qin et al. [QZG22], called *Quantifying blockchain extractable value: How dark is the forest?* relies on predefined DEXs, such as Uniswap V1/V2/V3, Sushiswap, Curve, and more.
This work focuses on detecting five types of MEV: Sandwich Attacks, Liquidations, Arbitrage, Clogging, and Transaction Replay. The authors aimed to find concrete examples of the first four types of MEV, but in the last case, Transaction Replay, they focused on transactions that could be replayed. This, however, means that these transactions would have been potential opportunities but not proven attacks.
For each type of MEV, a slightly different strategy was used. In their evaluation, they used publicly available blockchain data and applied heuristics to identify the concrete attack forms.

In the case of Sandwich attacks, the heuristics ensure

- that all three transactions are in the same block,
- that every Frontrunning transaction maps to exactly one Backrunning transaction,
- that the Frontrunning transactions swaps X to Y and the Backrunnning transaction Y to X,
- that the Frontrunning and Backrunning transactions are from the same user address or two different addresses, but the output of these transactions are channeled to the same Smart Contract
- that the amount of the assets sold in the Backrunning transaction is within 80 % and 120 % of the amount bought in the Frontrunning transaction.

Liquidations is the next form of MEV covered by Qin et al. [QZG22]. Their focus is on fixed-spread Liquidations, which allow liquidators to repay the discounted debt in exchange for the collateral. Identifying a Liquidation itself is easy since they are implemented as Smart Contract functions, and calls to them can be detected when analyzing transaction logs [13]. They differentiate between Frontrunning and Backrunning Liquidation strategies. A bot observes a Liquidation opportunity on a blockchain B_i in Frontrunning strategies. It uses Frontrunning to place their transaction before all other possible Liquidation transactions at block B_{i+1}. In Backunning strategies, the bot observes a transaction in the MemPool (private or public) that would create a Liquidation opportunity. The bot then tries to Backrun this transaction with a Liquidation transaction.
To classify the detected Liquidations in Frontrunning vs. Backrunning Liquidations, they use the observation that a Frontrunning Liquidation in block B_i acts on a Liquidation

[13] For example, Aave's LiquidationCall: https://docs.aave.com/developers/core-contracts/pool#liquidationcall

opportunity that, per definition, exists in block B_{i-1}. Therefore, they check for each Liquidation whether or not this position was liquidatable in the block B_{i-1}; if it was, it is classified as a Frontrunning - and if not, a Backrunning Liquidation.

The next investigated form of MEV is Arbitrage. As with Sandwich detection, heuristics were used to detect Arbitrage transactions. These Arbitrage transactions had to be included in a single transaction and, therefore, be atomic. They must have had more than one swap operation. Each swap operation must have used the output of the previous swap operation. In addition to that, the first swapped asset must have also been the last swapped asset. This heuristic is visualized in Figure 4.2.

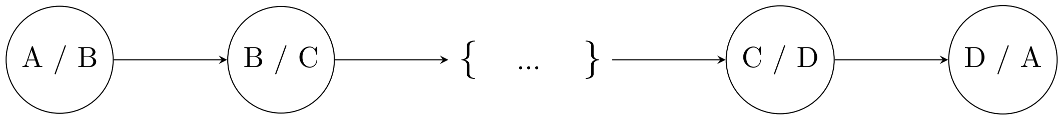


Figure 4.2: Illustration of the arbitrage detection heuristic used by Qin et al. [QZG22]. Each node indicates a swap operation.

The heuristics used by Qin et al. to detect Clogging are as follows. One address must consume more than 80 % of gas in at least five consecutive blocks.

The last form of MEV is transaction replay. To find replayable transactions, they tried to simulate a transaction replay of every transaction observed in the study timeframe. Simulating a transaction replay means duplicating this transaction and replacing the sender address with an adversary-controlled one. This modified transaction is then simulated locally and tested to see if it yielded a positive profit for the attacker. This technique does not require any insight into what the original transaction is doing since it only checks if the simulation yields a profit for the attacker. For each simulated transaction, Qin et al. had to download the block state of this transaction from an archival node to simulate it in the context of the transaction.

Additionally, Qin et al. used another heuristic to distinguish between public MemPool and private transactions. Private transactions are transactions that were sent to the proposer through a private collaboration channel. They identified private transactions by a gas price of 0. Usually, Ethereum clients do not relay such transactions to prevent DoS attacks; therefore, it is assumed that they have been submitted privately to the block builder.

Piet et al.

The work of Piet et al. [PFW22] in the paper *Extracting godl [sic] from the salt mines: Ethereum miners extracting value.* focuses on extracted MEV via private transactions. As the previous work presented here, this work also relies on known ABIs to understand

transactions. In this case, the ABI data is fetched from Etherscan [14] and 4byte [15].
On-chain data does not indicate whether a transaction was propagated over the public MemPool or submitted to the block builder over private channels. Therefore, to classify transactions into private or public MemPool transactions, the authors ran a modified Geth client [16], that locally stored all incoming transactions, along with their timestamp, hash, and source node. As the client receives a new block over the network, its transactions can be compared with the locally stored transactions. Therefore, whether a transaction was publicly known before being included in a block could be determined.
In addition to classifying private or public MemPool transactions, they quantified MEV in the observed dataset. This is done using a graph-based algorithm, in which the vertices are Ethereum addresses and the edges are internal transactions of one or more transactions. This algorithm classifies MEV transactions by identifying a cycle in this graph. This cycle has the following properties:

- Each internal transaction occurs in the cycle in the same order as in the transaction.
- Edges are connected with the same asset or part of a swap event.
- At least two different assets occur in the cycle.
- Each asset in the cycle must be part of a swap event.
- The start currency is the same as the end currency.

A cycle is, therefore, a transaction that uses swaps to profit on market inefficiencies. Characterizing these cycles into Arbitrage, Sandwich, or Backrunning is done using these cycles. If several assets are included, it is classified as arbitrage; if it is the same asset but spread on different DEX, it is Backrunning. If the cycle spans over multiple transactions, where the first transaction performs some swaps and the last transaction reverses them, then it is a Sandwich.

Torres et al.

The paper *Frontrunner Jones and the Raiders of the Dark Forest: An Empirical Study of Frontrunning on the Ethereum Blockchain* [TC+21] empirically studies three forms of MEV: Transaction Replay, Sandwich Attack, and Clogging.

This work focuses exclusively on Transaction Replay attacks involving MEV contracts in which both the attacker's transaction and the victim's transaction land on-chain. Due to the vast number of transactions in the blockchain, comparing all transactions would result in an impractical number of combinations. This work, therefore, split the blockchain into windows of 100 blocks, which were then analyzed in parallel. To further prefilter

[14]https://etherscan.io/, Accessed on 2024-04-27
[15]https://www.4byte.directory/, Accessed on 2024-04-27
[16]https://geth.ethereum.org/

the number of transactions, the input bytes of each transaction were split into n-grams of 4 bytes, then compared with all other transactions in a window to determine if 95 % of n-grams have been observed before in that window. To efficiently determine this, a bloom filter [Blo70] was used. A bloom filter is a probabilistic data structure that can tell if an element has been seen with a specific false positive rate but can confidently state that an element has not been seen before.

The following heuristics were then used to find two transactions in the prefiltered transactions that were part of a transaction replay attack, in which an attacker transaction T_A replaced a victim transaction T_V.

- The sender of T_A and T_V, as well as the receiver of T_A and T_V must be different.
- The replaced transaction T_V must have a lower gas price than the replacing transaction T_A.
- 25 % of the input bytes of T_V must match the input bytes of T_A

The last heuristic's low requirement originates from the assumption that MEV bots use contracts to perform the replacement attack and send additional control instructions to the Smart Contract in addition to the input of the victim's transaction. In addition to the mentioned heuristics, they identified a pair of transactions as part of a transaction replay attack if a simulated execution of T_V before T_A resulted in a different state than when executing T_A before T_V.

In the work of Torres et al., the detection of Sandwich attacks focuses solely on Sandwiches achieved by an attacker who sets the gas price of the Frontrunning transaction slightly higher and the gas price of the Backrunning transaction marginally lower than the victim's transaction. The focus lies exclusively on Sandwiches in AMM trading ERC-20 [17] tokens. The ERC-20 token standard defines transfer events that signal which tokens are transferred in what amount and in what direction. To classify a triple of transactions as a Sandwich attack, the Frontrunning transaction and the victim transaction must buy the same token, and the Backrunning transaction must sell this token to the AMM. Additionally, the amount of tokens bought in the Frontrunning transaction and the amount sold in the Backrunning transaction must be similar, with a one-percent allowed tolerance.

The third investigated MEV attack is the one of Clogging. The authors defined three strategies an attacker could use to consume the allowed gas in a block and hinder other transactions: controlled gas loop, uncontrolled gas loop, and assert. In the controlled gas loop strategy, a Smart Contract would execute instructions that consume gas in a loop. The attacker would control this loop by checking how much gas is left and eventually exiting the loop before the gas is consumed. In an uncontrolled gas loop strategy, the loop would run until the transaction runs out of gas and an out-of-gas exception is raised. In

[17]https://ethereum.org/de/developers/docs/standards/tokens/erc-20/

the assert strategy, an attacker would create a contract with a statement that evaluates to assert(false), which would consume all provided gas [18]. Under the assumption that an attacker would use a contract to perform a Clogging attack, the detection of these attacks starts with clustering transactions with the same receiver. All transactions in such a cluster must consume more than 21,000 gas; this condition was introduced to filter out transactions that do not execute code and instead transfer ether. A cluster of transactions must also use up to 99 % of the block gas limit. If such a cluster is detected, at least one neighboring block must contain it to classify it as a Clogging attack. Finally, the EVM instructions are analyzed to distinguish the different clogging strategies [19].

Weintraub et al.

The paper *A Flash(bot) in the Pan: Measuring Maximal Extractable Value in Private Pools* [WTNRS22] focuses on measuring the efficiency of Flashbots Auction [Fla]. In this analysis, the paper quantified three forms of MEV: Sandwich attacks, Arbitrage, and Liquidation.

In the analysis of Sandwiches, the heuristics from Torres et al. (section 4.2) were used and applied for transactions from the following exchanges: Bancor [20], SushiSwap [21], as well as Uniswap [22]. The profit was calculated by defining the costs as the transaction fees for the Front- and Backrunning transactions, as well as the miner bribers, and the revenue as the difference between the tokens bought in the Frontrunning transaction and the tokens sold in the Backrunning transaction.
In the case of Arbitrage, the author applied the heuristics developed by Qin et al. (section 4.2) for the following exchanges: 0x Protocol [23], Balancer [24], Bancor [25], Curve [26], SushiSwap [27] and Uniswap [28]. The costs of an arbitrage transaction are calculated by adding the transaction fees and the miner bribe, the revenue by calculating the assets that the transaction sender gained, including tokens and Ether.

Liquidations were detected by parsing transaction events for Liquidation events, such as

[18]This was true for Solidity before version 0.8.0 https://docs.soliditylang.org/en/latest/control-structures.html#error-handling-assert-require-revert-and-exceptions

[19]More than ten sequences of [GAS, GT, ISZERO, JUMPI] for a controlled gas loop.
More than ten sequences of [SLOAD, TIMESTAMP, ADD, SSTORE] in addition to an exception for an uncontrolled gas loop.
Checking a Panic Exception for the assert strategy.

[20]https://bancor.network/

[21]https://www.sushi.com/

[22]https://uniswap.org/

[23]https://0x.org/

[24]https://balancer.fi/

[25]https://bancor.network/

[26]https://curve.fi/#/ethereum/swap

[27]https://www.sushi.com/

[28]https://uniswap.org/

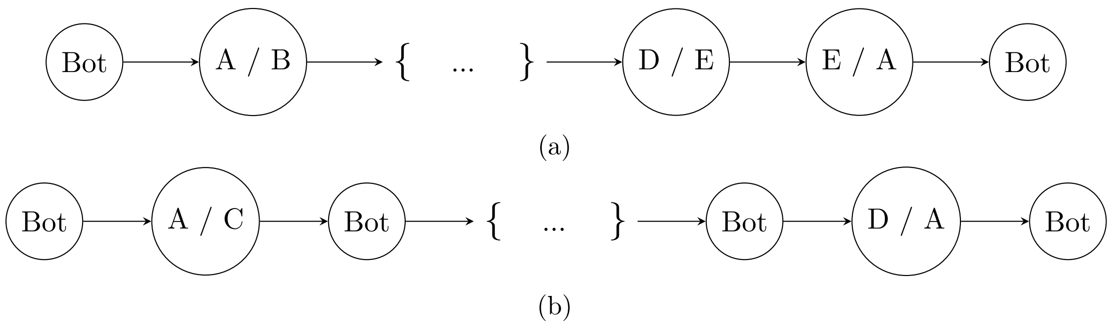


Figure 4.3: Illustration of the two types of arbitrage detection heuristics used by MEV-Inspect.
(a) illustrates an arbitrage transaction using fully routed swaps, in which the received token in a swap is directly used as an "in" token for the next swap
(b) illustrates an arbitrage transaction, where the tokens are always returned to the bot

Aave's *LiquidationCall* event [29] and Compounds's *LiquidateBorrow* event [30]. Based on these events, the collateral and liquidated debt were retrieved. The Liquidation events of the following platforms were parsed: Aave [31] and Compound [32]. The costs of such a Liquidation were calculated by adding the transaction fees, the miner payments, and the value of the liquidated debt. As revenue in the profit calculation, the value of the liquidated collateral was used.

MEV-Inspect

MEV-Inspect [33] is an open-source MEV analysis tool by flashbots [34]. It analyses two types of MEV: Arbitrage and Liquidation.

To do that, it uses transaction traces and known ABIs to classify them. The inspection starts by classifying swaps, identified by the knowledge of the known ABIs, to classify arbitrages. As defined by MEV-Inspect, an arbitrage starts with a bot selling a token from a decentralized exchange and ends with the same bot buying the token. In between lies a series of swaps with two strategies as seen in Figure 4.3.

The first heuristic in Figure 4.3 (a) is a fully routed swap. The second Figure 4.3 (b) is a heuristic in which the trades always return to the bot. According to MEV-Inspect [35], the first strategy Figure 4.3 (a) is the most common case, which occurs in over 99 % of cases. The profit of an arbitrage is calculated by taking the out amount of the last swap minus the in amount of the first swap. In addition to that, the miner bribe is also considered.

[29] https://docs.aave.com/developers/guides/liquidations
[30] https://docs.compound.finance/v2/ctokens/
[31] https://aave.com/
[32] https://compound.finance/
[33] https://github.com/flashbots/mev-inspect-py
[34] https://www.flashbots.net/
[35] https://github.com/flashbots/mev-inspect-py/blob/main/mev_inspect/arbitrages.py

Similar to the work of Weintraub et al. (section 4.2), Liquidations are detected by classifying the transaction trace logs. The profit of such Liquidations is calculated by subtracting the amount paid for the debt plus the miner bribe and transaction fees from the value of the purchased collateral.

Park et al.

The work of Park et al. in the paper *Unraveling the MEV enigma: Abi-free detection model using graph neural networks* focuses on MEV detection using Graph Neural Networks [PJL+23]. It detects Sandwiches and Arbitrages. For Sandwich detection, all transactions of a block are parsed, and Sandwich attacks are identified using the following heuristics:

- Both transactions have the same recipient.
- There must be at least two tokens traded in each transaction.
- All token transfers in the Frontrunning transaction must be negative (token bought)
- All tokens traded in the Backrunning transaction can not have a profit of 0 (either bought or sold)
- The tokens traded in the Frontrunning and Backrunning transaction must be the same.
- In the sum of all trades of the Front- and Backrunning transaction, the attacker must have increased their owned tokens.

Arbitrage detection begins by generating a token transfer graph and extracting 14 features for each node from the ERC-20 token transfer data (see Figure 4.4). These features capture various aspects of the transactions and addresses involved. The constructed token transfer graph is then processed by several Graph Neural Network (GNN) layers, specifically Graph Convolutional Networks (GCN), GraphSAGE, and Graph Attention Networks (GAT). These layers compute hidden states for the node features, effectively capturing the complex relationships and patterns within the token transfer data.

After the GNN layers compute the hidden states, a readout layer calculates the global mean of these hidden states across all nodes. This aggregated information is then passed through a linear layer that performs the final binary classification. The classification results are output to two nodes representing the MEV (Miner Extractable Value) and non-MEV classes.
The training data for this Arbitrage detection model was compiled by labeling Arbitrage transactions as '1' and non-Arbitrage transactions as '0'. Labeling an arbitrage transaction as '1' was based on five identified forms of Arbitrage.

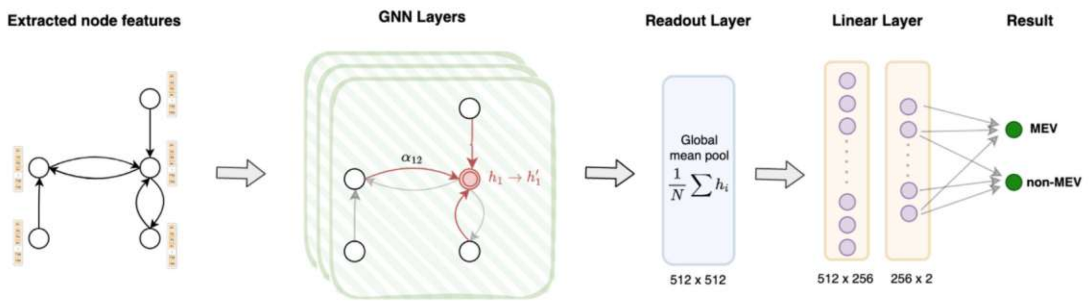


Figure 4.4: The structure of the GNN as used in the work of Park et al. [PJL+23]

EigenPhi

EigenPhi [Eig24a] is a data analysis firm specializing in MEV and liquidity data analysis. It provides a public website, `www.eigenphi.io`, that offers the visualization of MEV data. It offers a visualization of the transaction calls in a single transaction and estimates the MEV value. Figure 4.5 shows an example of a transaction visualized in EigenPhi. EigenPhi detects three forms of MEV: Sandwich, Arbitrage, and Liquidations.
To detect any of these types, it starts by filtering the transfers of a transaction [Eig24b]. Next, it calculates the balance changes of each involved contract or address. Revenue is then defined as the sum of all balance changes of the addresses or contracts belonging to the MEV searcher, typically identified by the "from" and "to" addresses. The corresponding exchange rates adjust these balance changes to get the total revenue of a transaction. Transaction costs are the sum of the builder's payment and transaction fees.
To our knowledge, EigenPhi has not published the concrete implementation of this algorithm.

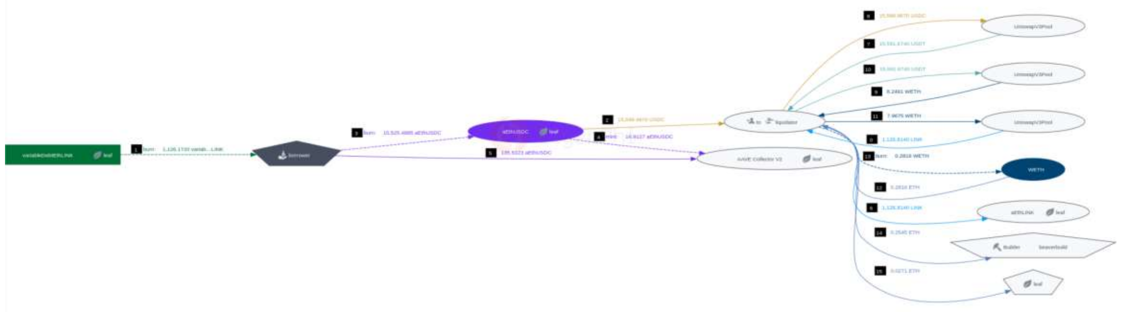


Figure 4.5: An example of a transaction visualization by EigenPhi.

Wunderlich et al.

Wunderlich et al. [Wun23] build their work on MEV-Inspect (see section 4.2). This work's Sandwich and Arbitrage detection method was based on existing approaches.

Sandwich attacks were analyzed using the heuristics by Torres et al. [TC+21], while the heuristics from Qin et al. [QZG22] were utilized for Arbitrage detection.
Liquidations were calculated over the transaction receipts and filtering Liquidation Events. Only Aave V1, V2, and Compound events were filtered for these Liquidation receipts. Once a Liquidation has been identified, its profit or loss is calculated by calculating the difference between debt and collateration.

Chi et al.

Chi et al.'s work, *Remeasuring the Arbitrage and Sandwich Attacks of Maximal Extractable Value in Ethereum,* [CHHW24] focuses on measuring Arbitrage and Sandwich attacks.

To detect Arbitrage transactions, they first identify swap events and initialize a graph based on all swaps in a transaction. If this graph includes a cycle, they check if this Arbitrage was profitable. If a graph does not include a cycle or is not profitable, it is not considered an Arbitrage.
Whether an Arbitrage is profitable or not is determined in the following way: First, ERC-20 Events are parsed for transfer events, and based on them, the balance changes for each type of token are calculated. It then aggregates all token changes and determines profit tokens as tokens where the gained amounts are higher than the lost amounts. If the swap ratio between the token-in and token-out amounts is too high, then this swap is not considered. Chi et al. argue that this step prevents errors in determining whether the transaction is profitable due to excessive conversion between tokens. For pairs with acceptable ratios, the algorithm simulates exchanging the lost token for the gained token using the directed graph representing token swaps. It checks if the gained token's balance is still positive after the exchange and adjusts the token balances based on the simulated exchanges. If all token balances are positive, then this algorithm considers a transaction a positive Arbitration transaction.
The next type of MEV quantified by Chi et al. is Sandwich attacks. Contrary to previous Sandwich identification mechanisms, this algorithm aims to detect Sandwich attacks that target multiple victim transactions. This is done by iterating over all possible pairs of transactions in a block where the first transaction's *from* address is the second transaction's *to* address. For all of these transactions, it calculates whether all balances of an attacker were positive, and if so, it labels it as a Sandwich attack.

4.2.1 Summary and Discussion

This review identified 16 significant works addressing MEV detection and quantification, including 14 papers or theses and two MEV detection tools.

Table 4.4 classifies the papers and tools mentioned in section 4.2.

Qin et al. [QZG22] analyzed the longest timeframe, spanning 32 months, while Li et al. [LZWD24] covered only 12 months and 28 hours.

Four papers use both on-chain and off-chain data for MEV analysis. In this analysis, ABIs and exchange rates were not classified as off-chain data.

The most analyzed MEV strategies are Sandwiches and Arbitrages. Conversely, only one paper examined PGAs, and two covered Replay attacks.

The strategies utilized by these works fall into two main categories: rule-based approaches and learning-based approaches. Most studies use rule-based MEV detection, including works by [DGK$^+$20], Qin et al. [QZG22], Piet et al. [PFW22], [TC$^+$21], Weintraub et al. [WTNRS22], EigenPhi [Eig24a], as well as MEV-Inspect [SHM21].

Rule-based MEV detection typically follows this process: first, transaction information from relevant blocks is extracted, and transaction logs are parsed. A graph is created, with nodes representing EOAs or Smart Contracts and edges representing transactions or sub-transactions. Concrete heuristics are applied to these graphs to identify the characteristics of MEV transactions. This method requires understanding various MEV strategies and high maintenance effort as new strategies must be incorporated into the detection mechanism. This can lead to false negatives if the strategies are unknown to the developer. Some studies in this review rely on a predefined set of ABIs and known decentralized exchange addresses to detect relevant transactions. The number of known ABIs in these studies ranges from 1 to 62, with up to 100,000 known DEX addresses. This increases maintenance efforts and the potential for false negatives, as transactions involving unknown decentralized exchanges may be missed.

McLaughlin et al. [MKV23] propose a rule-based detection mechanism that does not use predefined ABIs but relies solely on Transfer Events from the ERC-20 Token standard. This approach eliminates the need to store hard-coded AMM addresses and properties. Still, it introduces a new challenge: exchange aggregators like CowSwap [37] aggregate user transactions, which are not influenced by users' order. Relying only on Transfer Events in such transactions can lead to false positives.

The other approach for detecting and quantifying MEV is machine learning. This method requires creating a labeled dataset to train machine learning algorithms to detect MEV. Examples include Graph Neural Network (GNN) algorithms used by Yao et al. [YHL$^+$] and Park et al. [PJL$^+$23], and Cascading Classifiers by Liu et al. [LLPL24]. This strategy does not depend on known ABIs or addresses or requires understanding specific MEV strategies.

[37]https://docs.cow.fi/

Paper	Time of Study	Strategies						Rule-based/ Machine-based	Data Source	#o of ABIs (# of Exchanges)
		PGA	SW	ARB	LIQ	CLG	RPL			
[DGK^{+}20]	9 months	✓	✗	✗	✗	✗	✗	RB	On & Off	6 (-)
[QZG22]	32 months 2018-12-01 - 2021-08-05	✗	✓	✓	✓	✓	✓	RB	On	8 (60 830)
[PFW22]	12 days 2022-02-12 - 2022-02-24	✗	✓	✓	✗	✗	✗	RB	On & Off	n/a
[TC^{+}21]	1941 days 2015-07-30 - 2020-11-21	✗	✓	✗	✗	✓	✓	RB	On	4
[WTNRS22]	688 days 2020-05-04 - 2022-03-23	✗	✓	✓	✓	✗	✗	RB	On & Off	7
[SHM21]	n.a (Tool)	✗	✗	✓	✓	✗	✗	RB	On	12 (>75)
[Eig24a]	n.a (Tool)	✗	✓	✓	✓	✗	✗	RB	On	n/a
[Wun23]	365 days 2022-03-23 - 2023-03-23	✗	✓	✓	✓	✗	✗	RB	On	10
[PJL^{+}23]	928 days [36] 2020-05-04 - 2022-11-18	✗	✓	✓	✗	✗	✗	RB/ML	On & Off	62 (62)
[CHHW24]	89 Months July 2015 - August 2023	✗	✓	✓	✗	✗	✗	RB	On	n/a
[LLPL24]	1834 TXs (Training Set)	✗	✓	✗	✗	✗	✗	ML	On	n/a
[ZNW21]	2 367 980 blocks	✗	✓	✗	✗	✗	✗	RB	On	n/a
[YHL^{+}]	6 Months	✗	✓	✓	✓	✗	✗	ML	On	n/a
[Han22]	18 Months	✗	✗	✓	✗	✗	✗	RB	On	1
[LZWD24]	28h	✗	✓	✗	✗	✗	✗	RB	On	n/a
[MKV23]	28 months 2020-02-28 - 2022-07-10	✗	✗	✓	✗	✗	✗	RB	On	1 (100,000)

Table 4.4: Summary of all works covered in chapter 5. Abbreviations: **PGA** Priority Gas Auction, **SW** Sandwich, **ARB** Arbitrage, **LIQ** Liquidation, **CLG** Clogging, **RPL** Replay, **RB** Rule-Based, **ML** Machine Learning Based, **On** On-Chain Data, **Off** Off-Chain Data

CHAPTER 5

State-of-the-art MEV detection

This chapter aims to enhance the understanding of state-of-the-art detection methods by analyzing the output of selected MEV detection tools.

Not all papers mentioned in section 4.2 published their raw data, and among those that did, the study timeframes did not always overlap.

Therefore, we have chosen to compare the open-source tool MEV-Inspect [SHM21] with data from the EigenPhi [Eig24a] website. A key advantage of these tools is the flexibility to select our study timeframe freely.
This comparison reveals some unexpected differences in MEV-Inspect and EigenPhi detection capabilities. We present case studies of specific transactions, highlighting the discrepancies between the tools' results and comparing them with our manually calculated MEV values. Based on these case studies, we will enhance the detection capabilities of the open-source tool MEV-Inspect. Finally, we will introduce our heuristics for Liquidation detection, developed from the insights gained through the systematic review and analysis of these case studies.

5.0.1 Methodology and setup

Analyzing MEV transactions over the entire Ethereum blockchain requires significant resources and is beyond the scope of this thesis, we therefore decided to limit the study to the blocks 17980146[1] to 18580146 [2], or slot number 7166892 to 7771817 which consists of exactly 600,000 slots. To effectively compare the different datasets, we set up a PostgreSQL database to store all the data required for this study.

MEV-Inspect uses the block number as an argument for a function that calculates all forms of MEV for the given block. To calculate MEV, MEV-Inspect requires price

[1]https://etherscan.io/block/17980146
[2]https://etherscan.io/block/18580146

information on tokens and Ether. We used the CoinGeckoApi [3] to fetch this information. MEV-Inspect also requires an Ethereum node supporting transaction traces to analyze transactions. This study used a node from Grove [4] that supports the eth-trace module. MEV-Inspect allows the analysis of multiple blocks [5]. However, it turned out that our local hardware could not analyze all blocks at once because MEV-Inspect analyzes all blocks in a given range and stores all intermediate results in RAM before writing all results into the database at once. Since running the analysis on all 600,000 blocks was infeasible, we wrote a script that iterated over all 600,000 block numbers of our study blocks. This loop triggered the MEV-Inspect analysis in each iteration to analyze one block at a time. MEV-Inspect stores its results in a PostgreSQL database to be further analyzed. We then adapted MEV-Inspect to store its results in our already-set-up database to facilitate future comparisons. To fetch the EigenPhi Data for the timeframe, we used its API to fetch all MEV data. This API requires the transaction hash as an argument. To get this transaction hash, we queried a geth-node hosted at the University of Vienna in the version "Geth/v1.12.0-stable-e501b3b0/linux-amd64/go1.20.2". This geth-node provided all transaction hashes of a given block number. These transaction hashes were stored in a database so that this data could be used to fetch the data from EigenPhi. The calculated MEV values, profit, cost, revenue, and MEV type were then stored in the same PostgreSQL database as the MEV-Inspect calculations. This database then functioned as the basis for all analyses. For historical exchange rates between crypto assets and the USD, we are using the CoinGecko API [6].

5.1 Comparison MEV-Inspect and EigenPhi

In the timeframe, as mentioned in subsection 5.0.1, MEV-Inspect labeled 72,804 and EigenPhi 800,833 transactions. Figure 5.1 compares the number of labeled transactions by the MEV types Sandwiches, Arbitrages, and Liquidations.
As one can see from this graph, there is a considerable difference in the quantitative detection between these tools, with EigenPhi detecting ten times more transactions than MEV-Inspect. MEV-Inspect does not detect Sandwiches. Therefore, we can only compare Arbitrages and Liquidations. The most interesting difference is the number of Arbitrages detected, where MEV-Inspect detected 74,188 transactions, whereas EigenPhi detected 342,297 transactions, more than four times more. With the last category, Liquidations, EigenPhi (940 transactions) still counts more than double Liquidations compared to MEV-Inspect (426 transactions).
After analyzing the quantitative differences of the raw number of transactions, we continue to analyze the qualitative difference in the detection, meaning comparing the concrete differences in the calculated MEV profits.

[3]https://www.coingecko.com/ Accessed on 2024-08-01
[4]https://www.grove.city/ Accessed on 2024-06-13
[5]https://github.com/flashbots/mev-inspect-py?tab=readme-ov-file#inspect-many-blocks Accessed on 2024-06-13
[6]https://docs.coingecko.com/reference/coins-id-market-chart

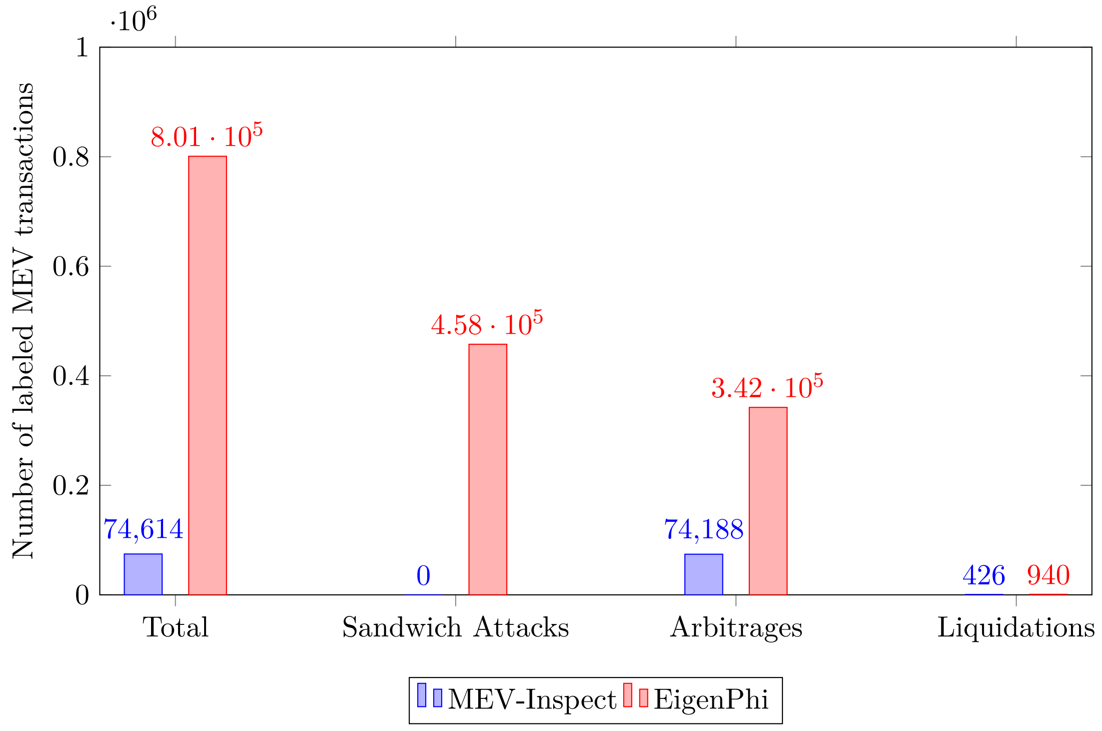


Figure 5.1: Comparison of the number of labeled MEV transactions between MEV-Inspect and EigenPhi in the block range of 79801461 to 18580146

Since MEV-Inspect does not detect Sandwich attacks, we will only compare MEV-Inspect with EigenPhi data from which the Sandwich attacks are excluded.

Figure 5.2 shows boxplots of the total calculated profits by MEV-Inspect and EigenPhi of blocks in the range of 79801461 to 18580146. This graph shows that EigenPhi analyzed a higher profit range for MEV transactions in this study timeframe. Figure 5.3 compares all transactions labeled as Arbitrage transactions. This plot looks very similar to the one of all transactions. This is not surprising since Arbitrages make up most of all labeled transactions both in MEV-Inspect and Eigenphi, as visualized in Figure 5.1 and therefore, the exclusion of the few Liquidation transactions has no real effect on the overall boxplot.

Figure 5.4 shows a more surprising difference in calculating Liquidations between the tools MEV-Inspect and EigenPhi. MEV-Inspect calculated a higher range of profits with the third quartile of 55.31 $ while EigenPhi had a third quartile of 5.03 $.

Negative MEV values Negative MEV values, as shown in Figure 5.2, Figure 5.3 and Figure 5.4, indicate that there exist transactions which result in a negative profit. This is particularly interesting since many forms of MEV can be executed atomically through Smart Contracts. One possible explanation for these negative results is that these transactions involve non-atomic types of MEV, such as Sandwich Attacks or Cross-

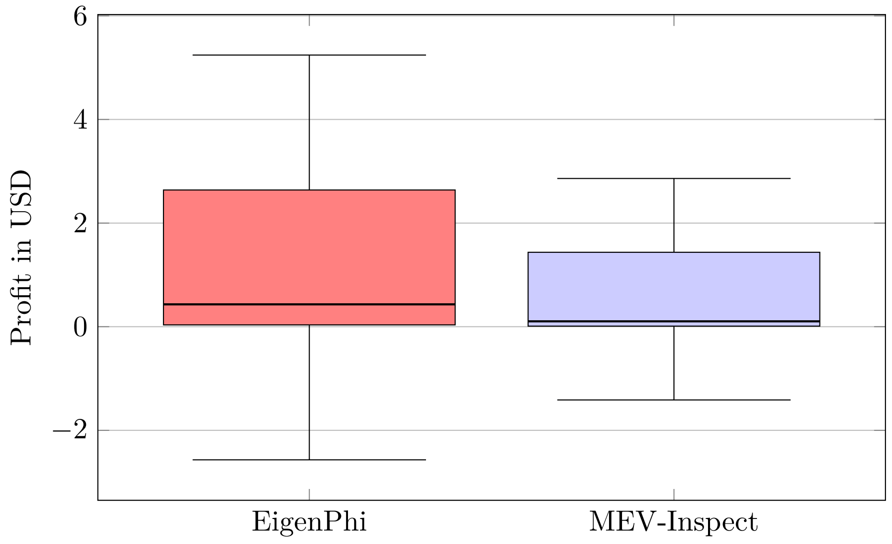


Figure 5.2: Comparison of the calculated MEV profits between MEV-Inspect and EigenPhi among all transactions in the block range 79801461 to 18580146, excluding Sandwich attacks from EigenPhi

Domain MEV. Alternatively, the negative reports could arise from strategies only partially detected by the heuristics used in MEV quantification and detection tools. While a detailed investigation of this phenomenon is beyond the scope of this thesis, it represents an interesting area for future research.

5.1.1 Manual transaction comparison of outliers

The differences in the results between those two tools motivate the following section, which involves manual transaction analysis, to understand the differences in the MEV calculation. For this reason, transactions were manually selected where the calculated profit between those two tools had the highest differences. Then, we manually investigated these transactions using the transaction call trace to understand the token flow and invoke contract calls.

After that, we manually calculated the MEV for each transaction and compared the results to the ones of MEV-Inspect and EigenPhi. Then, we tried to understand why the respective tools calculated what they calculated. We relied on code analysis for MEV-Inspect and tried to understand EigenPhi's results by reverse engineering its logic based on the calculated profit. In the following we will use $ to denote US Dollars.

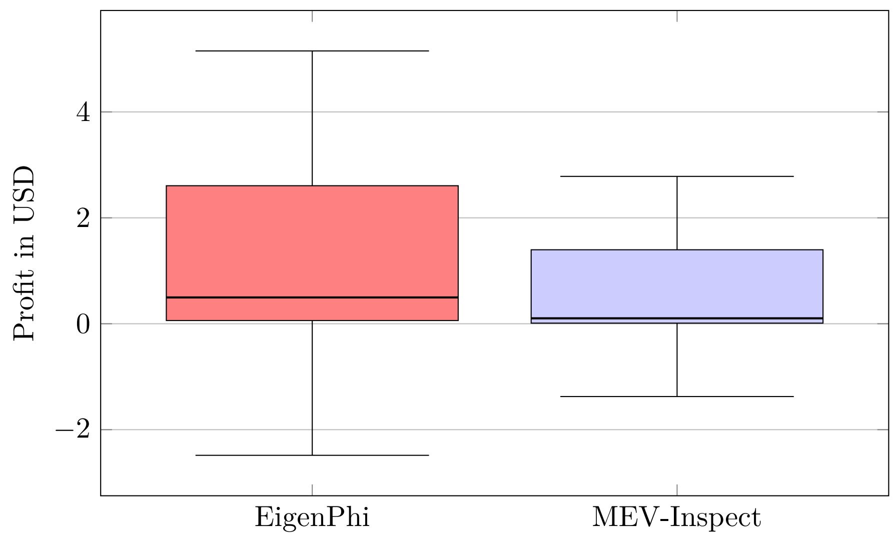


Figure 5.3: Comparison of the calculated MEV profits labeled as Arbitrage transactions between MEV-Inspect and EigenPhi, in the block range of 79801461 to 18580146

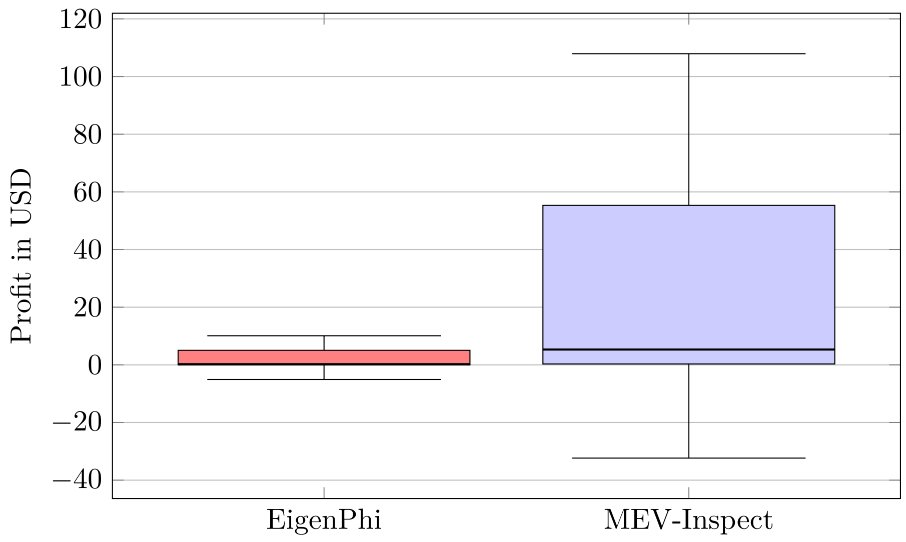


Figure 5.4: Comparison of the calculated MEV profits for Liquidations between MEV-Inspect and EigenPhi, in the block range of 79801461 to 18580146

Transaction 0x652...c4555 - Arbitrage

We begin the analysis with the transaction 0x652...c4555 [7] [8]. This transaction is classified as an Arbitrage transaction by both MEV-Inspect and EigenPhi. This transaction is

[7] https://etherscan.io/tx/0x652a043db7b712c011846cfdaf3468761e0689b6e93ec29ab215ad3582bc4555

[8] The following section will abbreviate the addresses to save space and increase readability.

interesting because of a very different profit calculation between these two tools, with MEV-Inspect reporting 297.72 $, while EigenPhi only reports a profit of 0.08 $.
This transaction starts with a call to the MEV contract 0x0000E0C...00000 [9]. This contract executes the Arbitrages by a total of four calls.
The first call is for the UniswapV3Pool 0xbcc...39295 [10] with the function *swap* [11] and the arguments as documented in Listing 5.1. The argument *zeroForOne* has the value 'true', which means that token0 is traded against token1. To understand what token0 and token1 are in the context of this Uniswap contract, one has to look at the contract creation, which happened in the transaction 0x90d...b7019 [12]. The details of the contract creation reveal that token0 is WETH [13], and token1 is the PAW token [14]. Therefore, this swap is a swap of WETH for PAW, with the searcher sending 0.177 WETH (*amountSpecified* parameter) to the AMM. The swapped 33,396,945,714.92 PAW tokens, however, are not returned to the searcher contract but forwarded to the recipient 0x0dba...37fdc9 [15], this swap recipient is specified by the parameter 'recipient'.

```
recipient = 0x0dba3dfee43d9b6450c716c58fdae8d3be37fdc9,
zeroForOne = true,
amountSpecified = 177258274143804789,
sqrtPriceLimitX96 = 3417834292187162269510808807191097 8,
data = 0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2dc63269ea166b70d
4780b3a11f5c825c2b761b0100271000000000000000000000000000000000000
000006be94e56eccec8e5d5ae3456
```

Listing 5.1: Arguments of the first swap call in transaction 0x652...c4555

The next call by the MEV contract 0x0000E0C...00000 is another *swap* with an Uniswap V2 AMM 0x428...347ed [16] with the arguments as documented in Listing 5.2 [17]. Like the previous AMM, this AMM trades a pair of PAW and WETH. In this swap, 0.15 WETH is swapped against 29,821,201,008.33 PAW. Similarly to the previous swap, the PAW tokens are sent to the recipient 0x0dba...37fdc9 [18].
This transaction ends with a call by the searcher contract 0x0000E0C...00000 to the contract 0x0dba...37fdc9 [19], which was the recipient of the first two swaps.
This contract is also an AMM that trades WETH and PAW tokens. This trade, however, is in the opposite direction. The 63,218,146,723.25 PAW tokens that were previously sent to this AMM were now swapped against a total of 0.34 WETH, which were sent back to

[9]https://etherscan.io/address/0x0000e0ca771e21bd00057f54a68c30d400000000
[10]https://etherscan.io/address/0xbcc489a50f0e2f09aec2d6f33ffe950cb6c39295
[11]https://docs.uniswap.org/contracts/v3/reference/core/UniswapV3Pool#swap
[12]https://etherscan.io/tx/0x90d4307904500dcc4262082a8969879e94f7158acfe25a4eb923598ed8bb7019
[13]https://etherscan.io/address/0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
[14]https://etherscan.io/address/0xDc63269eA166b70d4780b3A11F5C825C2b761B01
[15]https://etherscan.io/address/0x0dba3dfee43d9b6450c716c58fdae8d3be37fdc9
[16]https://etherscan.io/address/0x428b03ccd51ee4fcff7df6c7deae4139a4b347ed
[17]In the following we will omit the information in what transaction the AMM was created and only focus on the traded pairs
[18]https://etherscan.io/address/0x0dba3dfee43d9b6450c716c58fdae8d3be37fdc9
[19]https://etherscan.io/address/0x0dba3dfee43d9b6450c716c58fdae8d3be37fdc9

the MEV contract 0x0000E0C...00000.
To summarize this transaction, the MEV contract swaps 0,323 WETH against 63,218,146,723.25 PAW through two AMMs. These PAW tokens are then swapped against 0.344 WETH, which is transferred back to the MEV contract 0x0000E0C...00000. Therefore, the MEV contract has a revenue of 0.0211 WETH (0.344 WETH - 0.323 WETH).
For this day, the CoinGecko API returned a value of 1675.70 $ per WETH, which results in a raw revenue of 35.32 $. We must subtract all the gas fees from this raw revenue to get the lower profit bound. According to Etherscan [20], the transaction fee (Gas used * Gas Price) equals to 0.021 ETH. With an ETH price of 1679.12 $ of this day, this results in 35,31 $. Therefore, this transaction's profit is 0.013 $ (35,32 $ - 35,31 $).

```
amount0Out = 0, amount1Out = 29821201008326332378845249547,
to = 0x0dba3dfee43d9b6450c716c58fdae8d3be37fdc9, data = 0x
```

Listing 5.2: Arguments of the second swap call in transaction 0x652...c4555

This result is the closest to EigenPhi's result, which reports a profit of 0.083 $. A slightly different exchange rate might explain this difference compared to our manually calculated profit.
Next, we will analyze the difference between our manual and MEV-Inspect results.

MEV-Inspect noted this transaction as an Arbitrage MEV transaction with a start value of 0.146 WETH and an end amount of 0.344 WETH. The difference of this start and end value leads to a reported gross profit of 0.198 WETH, or 332.35 $. This start value explains the difference in the result, as the start value of 0.146 WETH was only one part of the invested amount by the MEV contract. The MEV contract additionally spends 0.1772 WETH by swapping it in the first call by this contract. Leaving out half of the swaps made in this transaction leads to MEV-Inspect overestimating this transaction's profit.
As mentioned in section 4.2, MEV-Inspect, when calculating the profit of Arbitrages, looks for fully routed swaps with the following strategy: BOT -> A/B -> B/C -> C/A -> BOT [21]. This arbitrage, however, follows this strategy: BOT -> A/B, BOT -> A/B -> B->A, B->A, A->BOT.

Transaction 0x98f...3f107 - Arbitrage

Contrary to the previously analyzed transaction, for this transaction, EigenPhi reports a higher transaction profit than MEV-Inspect. This transaction, 0x98f...3f107 [22], has a reported profit by EigenPhi of 60.83 $ and only 1.54 $ by MEV-Inspect.

[20] https://etherscan.io/tx/0x652a043db7b712c011846cfdaf3468761e0689b6e93ec29ab215ad3582bc4555
[21] https://github.com/flashbots/mev-inspect-py/blob/b3438d73536412c39f7b8dc64660b992e5f9b206/mev_inspect/arbitrages.py#L34
[22] https://etherscan.io/tx/0x98f6b1c48091e1a1e6c21411c34f76f648a1c30195dcffa80bd5a9303dd3f107

The transaction sender is the address 0x425...88959 [23], which starts this transaction by invoking the MEV contract 0xca8...e08dc [24].

This contract triggers a swap call to the UniswapV3Pool 0x11b...697f6 [25]. The arguments for this swap call are listed in Listing 5.3. In this swap, the MEV contract sends 7.417 WETH to the AMM, which returns 11,871.84 USDT to the MEV contract 0xca8...e08dc. When calling a Uniswap V3 swap call, the sender must implement the *IUniswapV3SwapCallback* [26] interface, which provides a function *uniswapV3SwapCallback*, which the AMM triggers after sending their token to the defined recipient. When the call for this function ends, the token owed by the sender must be sent to the AMM; otherwise, the transaction will revert. In this *uniswapV3SwapCallback* function the contract 0xca8...e08dc triggers another swap call of SolidlyV3Pool - 0x3198...5ed7e [27], in which 11,812.11 USDT are sent from 0xca8...e08dc in exchange of 7.422 WETH.

The final call of this transaction is by the MEV contract 0xca8...e08dc to the withdraw function of the WETH [28] contract, withdrawing 0.0056 Ether.
To summarize this transaction, there were two swaps performed, one with UniswapV3Pool [29] and one to SolidlyV3Pool [30]. First, 7.417 WETH were swapped against 11,871.84 USDT, and then, in the next call, only 11,821.11 USDT were sent to be swapped against 7.422 WETH.
To calculate the profit of this transaction, we have to investigate what the MEV contract gains by this Arbitrage. As mentioned, it gained the ETH of 0.0056 Ether, but it also gained 59.73 USDT. This is the difference between the USDT received by the first swap and the USDT spent in the second swap. When considering the fee paid to the miner of 7.12 $, we define the profit of this arbitrage as 61.28 $.
Again, this profit is so close to the profit calculated by EigenPhi (60.83 $) that we assume that different exchange rates best explain the difference.
MEV-Inspect, on the other hand, reported a much lower profit in the amount of 1.54 $. The gained WETH, which was later withdrawn to ETH, adjusted by the exchange rate of this day, amounts to exactly 8.659 $. Subtracting the miner's payment of 7.12 $ results in 1.539 $, which is the MEV profit returned by MEV-Inspect. This leads to the conclusion that MEV-Inspect only calculated the gain in ETH while failing to consider the USDT that the MEV contract gained by this transaction.

```
recipient = 0xca8acfdcee7531be980e7670d9e6b80b8c2e08dc, zeroForOne = true,
amountSpecified = 7416920663287467259, sqrtPriceLimitX96 = 4295128740
```

Listing 5.3: Arguments for the first swap call of transaction 0x98f...3f107. The data argument of the transaction was removed for brevity.

[23]https://etherscan.io/address/0x425a4a539c085ff2568e19ee1304e97a92688959
[24]https://etherscan.io/address/0xca8acfdcee7531be980e7670d9e6b80b8c2e08dc
[25]https://etherscan.io/address/0x11b815efb8f581194ae79006d24e0d814b7697f6
[26]https://docs.uniswap.org/contracts/v3/reference/core/interfaces/callback/IUniswapV3SwapCallback
[27]https://etherscan.io/address/0x3198eadb777b9b6c789bfc89491774644e95ed7e
[28]0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
[29]https://etherscan.io/address/0x11b815efb8f581194ae79006d24e0d814b7697f6
[30]https://etherscan.io/address/0x3198eadb777b9b6c789bfc89491774644e95ed7e

Transaction 0x419...6b03a - Liquidation

The next transaction that is manually investigated is the transaction 0x419...6b03a [31]. The calculated MEV value differs greatly between the two compared tools, EigenPhi and MEV-Inspect. While EigenPhi reports a profit of 167.09 $, MEV-Inspect reports only 1.87 $.
This transaction was sent by 0x200...00002 [32] and started with a call to the MEV contract 0xcbed...df827 [33]. This contract starts taking a flash loan from 0xba1...bf2c8 [34], Balancer Vault of 166.33 USDC [35], without any fees. Any recipient who calls this contract's flashloan call must implement the *IFlashloanRecipient* interface, which provides the *receiveFlashLoan* function. The Balancer Vault calls this *receiveFlashLoan* function, which triggers further calls by the MEV contract 0xcbed...df827.
This contract approves 166.33 USDC via a Proxy contract [36] for the Aave lending protocol 0x793...bffcb [37]. This is the ERC20 *approve* function [38], which allows the Aave lending protocol 0x793...bffcb to spend the specified amount.
The next call is the *liquidationCall* [39]of the Aave Protocol 0x793...bffcb. The collateral is WETH; the debt is USDC, the borrower 0x26b...23076 [40], and the debt to cover is 166.33 USDC. The MEV contract receives 0.1066 WETH for this Liquidation in exchange for the debt of 166.33 USDC.
Next to this call, the searcher swaps the 0.1066 WETH to 0.09933 wstETH (Wrapped liquid staked Ether 2.0 - 0x7f3...e2ca0 [41]).
Next, a swap of this wstETH in exchange for 174.09 USDC on a Uniswap V3 AMM follows.
The searcher then pays back the flash loan of 166.33 USDC.
After the flash loan's payback, the transaction performs two more swaps: 7.754 USDC to 7.749 USDT and 7.749 USDT to 0.004748 WETH. The transaction ends with the withdrawal of this 0.004748 WETH by the searcher contract and sending 0.004748 ETH to the contract 0x00000639...e4df700000 [42] via the fallback function [43].
Since this last contract 0x00000639...e4df700000 only appears at the end of this transaction and does not trigger any further calls, we assume that the original transaction sender controls this contract. This assumption is also strengthened by the fact that this contract also interacts with the searcher contract 0xcbed...df827 at other transactions, for example,

[31]https://etherscan.io/tx/0x41909f70701c5126aaf74fe09c2e9d4b5e59441a53435aa337ea0a4c08c6b03a
[32]https://etherscan.io/address/0x2000043a2d343a172bd34929bc308a89ab000002
[33]https://etherscan.io/address/0xcbed8b1b92c6e4f86a58f66da7ba26a7172df827
[34]https://etherscan.io/address/0xBA12222222228d8Ba445958a75a0704d566BF2C8
[35]The USDC contract has six decimals: https://etherscan.io/token/0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48
[36]0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48
[37]https://etherscan.io/address/0x7937d4799803fbbe595ed57278bc4ca21f3bffcb
[38]https://eips.ethereum.org/EIPS/eip-20
[39]https://docs.aave.com/developers/core-contracts/pool#liquidationcall
[40]https://etherscan.io/address/0x26b63e5c83238c2f614d287252e61b0398123076
[41]https://etherscan.io/address/0x7f39c581f595b53c5cb19bd0b3f8da6c935e2ca0
[42]https://etherscan.io/address/0x00000639caea2f4991b946c1f68686e4df700000
[43]https://solidity-by-example.org/fallback/

in transaction 0x339...7185a [44].
The first approach to calculating this transaction's profit is to calculate the difference between the gained collateral and the paid-back flash loan. From the received collateral of 0.1066 WETH, which eventually was swapped to 174.09 USDC, we subtract the amount paid back to the flash loan lender, which is 166.33 USDC, resulting in a profit of 7.76 USDC. The costs of this transaction can be quantified with the transaction fee of 0.00434 Ether, or 7.15 $.

Based on the reported values of the two tools, one can assume that EigenPhi reports only the collateral gained by the Liquidation and MEV-Inspect the Liquidation's profits minus the flash loan.

Transaction 0xb0d...0faa5 - Liquidation

The following transaction analyzed in this chapter is the Liquidation transaction with the transaction hash 0xb0d...0faa5 [45]. As opposed to the previously analyzed Liquidation, this transaction has a higher reported profit in MEV-Inspect, 47,986.21 $, as opposed to the 3,715.65 $ as reported by EigenPhi.
The transaction is triggered by the address 0x3b2...9b655 [46], with a call to the MEV contract 0x80d...72a13 [47], which delegates the logic to the contract 0x455...ddd81 [48].
This contract starts with a swap, exchanging 410,311.09 USDC against 12.21 Wrapped BTC with the uniswap contract 0x99ac...abc35 [49].
After that, the contract triggers the Liquidation call on the Aave Lending Pool 0x7d2...dc7a9 [50]. The collateral of this liquidated debt is USDC, the debt WBTC, and the debt to cover is 12.21 WBTC.
For this debt, the searcher contract pays this 12.21 WBTC and receives 414,104.93 USDC. The Liquidation call was the last in this transaction.
To sum up this transaction, the Liquidation itself was an exchange of WBTC valued 366,338.81 $ in exchange of USDC valued 414,363.44 $ [51]. The difference in value between the debt and the received collateral is, therefore, 48,024.63$.
However, the searcher contract did not start with the WBTC used in the Liquidation; it had to swap USDC to gain WBTC. Therefore, the more accurate view of the profit is to consider the USDC gain that the MEV contract made. It started with 410,311.09 USDC and ended with 414,104.93 USDC, gaining 3,793,84 USDC or 3,796.21$.
To calculate the costs of this transaction, we take the transaction fee of 0.023 Ether, which is equivalent to 38.42 $ at an exchange rate of 1664.57. Altogether, this transaction gained a profit of 3796.21 $ - 38.42 $ = 3,757.79 $.

[44] https://etherscan.io/tx/0x33930adf996583e430e5be336de5c4db05a8f41258a83189d3e192179917185a
[45] https://etherscan.io/tx/0xb0d1f0a56488b3dc6cd3cb3a4f860ca4fc83f2b30a84869ab92702360450faa5
[46] https://etherscan.io/address/0x3b2d2afeaff10f05ebeba4a6c14dff787fd9b655
[47] https://etherscan.io/address/0x80d4230c0a68fc59cb264329d3a717fcaa472a13
[48] https://etherscan.io/address/0x45576a1fba3b5f38b43232a720aec7482ecddd81
[49] https://etherscan.io/address/0x99ac8ca7087fa4a2a1fb6357269965a2014abc35
[50] https://etherscan.io/address/0x7d2768de32b0b80b7a3454c06bdac94a69ddc7a9
[51] Exchange rate of USDC on 2023-10-23: 1.000624272752208

The result of EigenPhi is much closer to our calculated profit than MEV-Inspect's result. MEV-Inspect reported 47,986.21 $, which is the profit made by the Liquidation itself (48,024.63 $) minus the transaction costs of 38.42 $.
Therefore, and this is supported by the code analysis, especially with the MEV-Inspect's Liquidation query in Listing 1, we conclude that MEV-Inspect only considered the value gained by the Liquidation itself. However, it failed to consider the MEV contract's costs to retrieve the correct token amount to liquidate this debt.

Transaction 0x350...d640c - Liquidation

The last Liquidation transaction analyzed is the transaction 0x350...d640c [52]. EigenPhi quantifies the MEV value of this transaction with a profit of 19.32 $ [53], while MEV-Inspect quantifies it with 405.24 $.
This transaction starts with a call from 0x41d...daa1c [54] to the contract 0xb20...da1aa [55]. After that, the first swap happens, where the searcher contract swaps 1126.814 Chainlink (LINK) Token in exchange for 7.9675 WETH.
Then follows the Liquidation call to the Aave Pool 0x878...fa4e2 [56]. The liquidated debt is 1,126.8140 LINK in exchange for 15,598.967 USDC collateral.
Two swaps follow this Liquidation: First, swapping the 15,598.967 USDC against 15,591.674 USDT and then this 15,591.674 USDT in exchange for 8.2491 WETH.
The revenue of the Liquidation itself is 15,606.94 $ (15,598.967 USDC) - 14,689.52 $ (1,126.8140 LINK) = 917.42 $. When considering the swaps before and after the transaction, we observe that the contract started sending 7.9675 WETH and received 8.2491 WETH in the last swap. This difference of 0.282 WETH was valued at 531.4667 $.
The costs of this transaction were valued at 512.178 $. This includes the transaction fees for the gas, as well as a direct miner payment of 0.2545 ETH to the block builder beaverbuild [57].
To conclude, we would estimate the transaction MEV profit at 19.289 $.
This result, again, is closest to the profit calculated by EigenPhi. Our conclusion, in terms of MEV-Inspect's calculation, is the same as with the previous transaction: MEV-Inspect only calculated the profit from the transaction itself but failed to consider the swaps made before and after the Liquidation.

5.2 Improving found inconsistencies in MEV-Inspect

In section 5.1, we observed a noticeable difference in the results of the two tools EigenPhi and MEV-Inspect. In the following subsection 5.1.1, we've analyzed handpicked transac-

[52]https://etherscan.io/tx/0x350ca80a3db7d792d37a200933d88a3dfd472cfdd2cd11f127d73f26767d640c
[53]https://eigenphi.io/mev/ethereum/liquidation/tx/0x350ca80a3db7d792d37a200933d88a3dfd472cfdd2cd11f127d73f26767d640c
[54]https://etherscan.io/address/0x41d32ac20ceb0441cab159444af1acc47f3daa1c
[55]https://etherscan.io/address/0xb206ebe579be55f5b57119bb2e7cc63708eda1aa
[56]https://etherscan.io/address/0x87870bca3f3fd6335c3f4ce8392d69350b4fa4e2
[57]https://etherscan.io/address/0x95222290dd7278aa3ddd389cc1e1d165cc4bafe5

tions with a substantial difference in the calculated MEV.
In this section, we focus on improving some of the inconsistencies found in MEV-Inspect. Since MEV-Inspect is an open-source tool and, to the best of our knowledge, EigenPhi has not made its source code and algorithms public, we focus on improving MEV-Inspect. We start by analyzing MEV-Inspect's source code and try to understand the profit calculation by following the execution paths of concrete transactions. We have found one concrete shortcoming in MEV-Inspect, for which we offer a concrete solution.
Later in this chapter, we will define a new Liquidation heuristic that includes the learnings from subsection 5.1.1. We will offer concrete changes to the MEV-Inspect code base, including this new heuristic.

5.2.1 Missing Padding

First, we examine the last Liquidation transaction, 0xb0d...0faa5, analyzed in section 5.1.1, which is the simplest because it has a low number of calls.
MEV-Inspect begins every analysis of a transaction by classifying its traces. This classification is based on stored ABIs and a concrete classification strategy for each ABI. A classification of the Uniswap mechanism, which is called in this transaction, already exists. However, the hardcoded mapping between ABI and contract addresses did not include the Uniswap contract address 0x99ac...abc35. Therefore, the first step to fix this transaction analysis was to include the address 0x99ac...abc35 to the valid contract addresses for Uniswap.
After adding the contract address to the known addresses, it did not, however, lead to the detection of the swap call. A deep investigation of the MEV-Inspect Code led to the detection of the root cause of the problem. MEV-Inspect uses the contract ABI of a function to decode the Calls' Calldata.
Calldata is a hexadecimal encoded string that resembles the function of a call and its parameters. The swap call that was not recognized by MEV-Inspect has the following call data:

```
0x128acb0800000000000000000000000080d4230c0a68fc59cb264329d3a717
fcaa472a13000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
5f88725734000000000000000000000000fffd8963efd1fc6a506488495d951d
5263988d25000000000000000000000000000000000000000000000000000000
00000000a0000000000000000000000000000000000000000000000000000000
000000002ba0b86991c6218b36c1d19d4a2e9eb0ce3606eb482260fac5e5542a
773aa44fbcfedf7c193bc2c599000bb8
```

After the '0x' hexadecimal identifier, 128acb08, the first four bytes represent the function selector. Following are the encoded function call values of the types [address, bool, int256, uint160, bytes]. However, the last parameter, which represents the data parameter of the

swap call [58] is of type bytes.
The library that MEV-Inspect uses for decoding the calldata is eth-abi. This library failed to decode the calldata, since the last parameter was not padded to a multiple of 32 [59]. In this case, the fix was to pad the last parameter with 0s until its length was a multiple of 32. This fix led to the swap being recognized by MEV-Inspect.

5.2.2 Including swaps for the Liquidation Calculation

Both Liquidation transactions 0xb0d...0faa5 and 0x350...d640c analyzed in section 5.1.1 have a common fault in MEV-Inspect: They both calculate the MEV by including only the value difference of the debt and collateral.
Listing 1 shows the query used to calculate the MEV-value of all Liquidations.

Listing 1: The query used by MEV-Inspect to calculate the MEV of a Liquidation. Commit: `https://github.com/flashbots/mev-inspect-py/commit/b3438d73536412c39f7b8dc64660b992e5f9b206`.

```

INSERT INTO mev_summary (
    SELECT
        NULL,
        l.block_number,
        b.block_timestamp,
        l.protocol as protocol,
        l.transaction_hash,
        'liquidation' as type,
        l.received_amount*
        (
            SELECT usd_price
            FROM prices
            WHERE token_address = l.received_token_address
            AND timestamp <= b.block_timestamp
            ORDER BY timestamp DESC
            LIMIT 1
        )
        /POWER(10, received_token.decimals)

        -

        l.debt_purchase_amount*
        (
```

[58] https://etherscan.io/address/0x99ac8ca7087fa4a2a1fb6357269965a2014abc35#code
[59] https://docs.soliditylang.org/en/v0.8.26/abi-spec.html

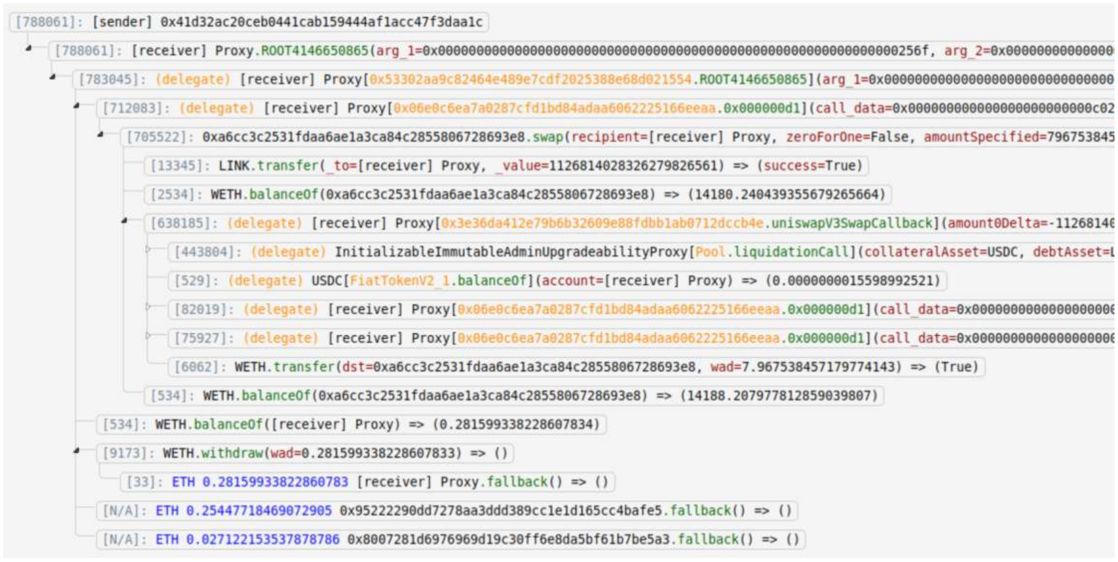
```
            SELECT usd_price
            FROM prices
            WHERE token_address = l.debt_token_address
            AND timestamp <= b.block_timestamp
            ORDER BY timestamp DESC
            LIMIT 1
        )
        /POWER(10, debt_token.decimals) as gross_profit_usd,
        (
            (
                ((mp.gas_used * mp.gas_price) + mp.coinbase_transfer) /
                POWER(10, 18)
            ) *
            (
                SELECT usd_price
                FROM prices p
                WHERE
                    p.timestamp <= b.block_timestamp
                    AND p.token_address = '0xeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee'
                ORDER BY p.timestamp DESC
                LIMIT 1
            )
        ) AS miner_payment_usd,
        mp.gas_used,
        mp.gas_price,
        mp.coinbase_transfer,
        mp.gas_price_with_coinbase_transfer,
        mp.miner_address,
        mp.base_fee_per_gas,
        ct.error as error,
        ARRAY[l.protocol]
    FROM liquidations l
    JOIN blocks b ON b.block_number = l.block_number
    JOIN tokens received_token
        ON received_token.token_address = l.received_token_address
    JOIN tokens debt_token
        ON debt_token.token_address = l.debt_token_address
    JOIN miner_payments mp ON
        mp.block_number = l.block_number AND
        mp.transaction_hash = l.transaction_hash
    JOIN classified_traces ct ON
        ct.block_number = l.block_number AND
        ct.transaction_hash = l.transaction_hash
    WHERE
        b.block_number >= :after_block_number AND
        b.block_number < :before_block_number AND
        ct.trace_address = '{}' AND
        l.debt_purchase_amount > 0 AND
        l.received_amount > 0 AND
        l.debt_purchase_amount <
         ↪  115792089237316195423570985008687
        907853269984665640564039457584007913129639935
)
```

This query does not consider any swaps; it calculates the MEV by subtracting the debt's value from the received collateral's value. However, the analyses of both these transactions show that omitting the swaps in the calculation can lead to big differences in the calculated MEV value.
Therefore, to improve the MEV calculation by MEV-Inspect, we aim to include the token swaps before or after the Liquidation.
We define a new heuristic for the Liquidation's value calculation. This new heuristic requires the exact localization of each call in a transaction. This localization is done using the trace address of each call.
The trace address is included in the trace module of the Ethereum JSON RPC [60]. It allows the exact localization of each call in the transaction trace, including the precise localization of recursive calls. Figure 5.5 shows a screenshot of the transaction order of transaction 0x350...d640c [61] made with the tool EthTx [Eth21], which visualizes this transaction trace. The trace begins with the transaction's sender, who begins the transaction by calling the contract 0xb20...da1aa. Each line represents a contract call, internal function execution, or emitted event. The structure of this visualization includes the following information:

- Call Order: The order of the lines represents the chronological sequence of calls and events
- Call Hierarchy: The Indentation level represents the hierarchical relationship of calls. A sub-call has an increased indentation level compared to its parent call.

This visualization was derived from the raw transaction trace provided by the Ethereum JSON RPC trace module. An array of integers represents this raw transaction trace in

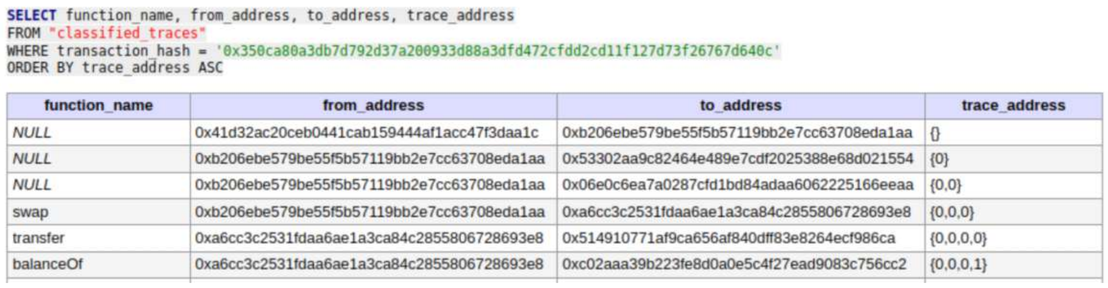
[60]https://openethereum.github.io/JSONRPC-trace-module
[61]https://etherscan.io/tx/0x350ca80a3db7d792d37a200933d88a3dfd472cfdd2cd11f127d73f26767d640c



```
[788061]: [sender] 0x41d32ac20ceb0441cab159444af1acc47f3daa1c
  [788061]: [receiver] Proxy.ROOT4146650865(arg_1=0x000000000000000000000000000000000000000000000000000000000000256f, arg_2=0x00000000000000
    [783045]: (delegate) [receiver] Proxy[0x53302aa9c82464e489e7cdf2025388e68d021554.ROOT4146650865](arg_1=0x0000000000000000000000000000000
      [712083]: (delegate) [receiver] Proxy[0x06e0c6ea7a0287cfd1bd84adaa6062225166eeaa.0x000000d1](call_data=0x000000000000000000000000c02
        [705522]: 0xa6cc3c2531fdaa6ae1a3ca84c2855806728693e8.swap(recipient=[receiver] Proxy, zeroForOne=False, amountSpecified=796753845
          [13345]: LINK.transfer(_to=[receiver] Proxy, _value=1126814028326279826561) => (success=True)
          [2534]: WETH.balanceOf(0xa6cc3c2531fdaa6ae1a3ca84c2855806728693e8) => (14180.240439355679265664)
          [638185]: (delegate) [receiver] Proxy[0x3e36da412e79b6b32609e88fdbb1ab0712dccb4e.uniswapV3SwapCallback](amount0Delta=-1126814
            [443804]: (delegate) InitializableImmutableAdminUpgradeabilityProxy[Pool.liquidationCall](collateralAsset=USDC, debtAsset=L
            [529]: (delegate) USDC[FiatTokenV2_1.balanceOf](account=[receiver] Proxy) => (0.000000015598992521)
            [82019]: (delegate) [receiver] Proxy[0x06e0c6ea7a0287cfd1bd84adaa6062225166eeaa.0x000000d1](call_data=0x000000000000000000
            [75927]: (delegate) [receiver] Proxy[0x06e0c6ea7a0287cfd1bd84adaa6062225166eeaa.0x000000d1](call_data=0x000000000000000000
            [6062]: WETH.transfer(dst=0xa6cc3c2531fdaa6ae1a3ca84c2855806728693e8, wad=7.967538457179774143) => (True)
          [534]: WETH.balanceOf(0xa6cc3c2531fdaa6ae1a3ca84c2855806728693e8) => (14188.207977812859039807)
      [534]: WETH.balanceOf([receiver] Proxy) => (0.281599338228607834)
      [9173]: WETH.withdraw(wad=0.281599338228607833) => ()
        [33]: ETH 0.28159933822860783 [receiver] Proxy.fallback() => ()
      [N/A]: ETH 0.25447718469072905 0x95222290dd7278aa3ddd389cc1e1d165cc4bafe5.fallback() => ()
      [N/A]: ETH 0.027122153537878786 0x8007281d6976969d19c30ff6e8da5bf61b7be5a3.fallback() => ()
```

Figure 5.5: Screenshot of the execution trace of the transaction 0x350...d640c [Eth21]

the data provided by the RPC trace module. Figure 5.6 shows the first few lines of the raw execution trace of the same transaction 0x350...d640c.



```sql
SELECT function_name, from_address, to_address, trace_address
FROM "classified_traces"
WHERE transaction_hash = '0x350ca80a3db7d792d37a200933d88a3dfd472cfdd2cd11f127d73f26767d640c'
ORDER BY trace_address ASC
```

function_name	from_address	to_address	trace_address
NULL	0x41d32ac20ceb0441cab159444af1acc47f3daa1c	0xb206ebe579be55f5b57119bb2e7cc63708eda1aa	{}
NULL	0xb206ebe579be55f5b57119bb2e7cc63708eda1aa	0x53302aa9c82464e489e7cdf2025388e68d021554	{0}
NULL	0xb206ebe579be55f5b57119bb2e7cc63708eda1aa	0x06e0c6ea7a0287cfd1bd84adaa6062225166eeaa	{0,0}
swap	0xb206ebe579be55f5b57119bb2e7cc63708eda1aa	0xa6cc3c2531fdaa6ae1a3ca84c2855806728693e8	{0,0,0}
transfer	0xa6cc3c2531fdaa6ae1a3ca84c2855806728693e8	0x514910771af9ca656af840dff83e8264ecf986ca	{0,0,0,0}
balanceOf	0xa6cc3c2531fdaa6ae1a3ca84c2855806728693e8	0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2	{0,0,0,1}

Figure 5.6: The beginning of the raw execution trace of 0x350...d640c

The column *trace_address* represents this execution trace. The integer value represents the call order. For example, the last line has the value of {0,0,0,1}, while the line before has the value of {0,0,0,0}. The increased integer value of the last line signals that this line was called after the line before.

The length of the array represents the call hierarchy. Each new sub call increases the length of this array, as seen in the first two lines. The first call, from the transaction sender to the contract, has an empty array {}. The following subcall increases the array length to {0}.

This trace feature allows the ordering of transaction calls, which is essential for our newly defined Liquidation calculation heuristic. This heuristic is defined as follows:
It starts with the ordering of all calls of a transaction in the order according to the

trace_address. We continue filtering all swap and Liquidation events with these ordered transaction calls. Each swap event has an "in" value and an "out" value. The "in" value is the number of tokens sent to the AMM, and the "out" value is the number of tokens received from the AMM. In this terminology, a Liquidation's "in" value is the debt sent to the Lending contract, and the "out" value is the collateral received. Figure 5.7 visualizes this flow of tokens for the transaction 0x350...d640c, Figure 5.8 visualizes the simpler token flow of transaction 0xb0d...0faa5.

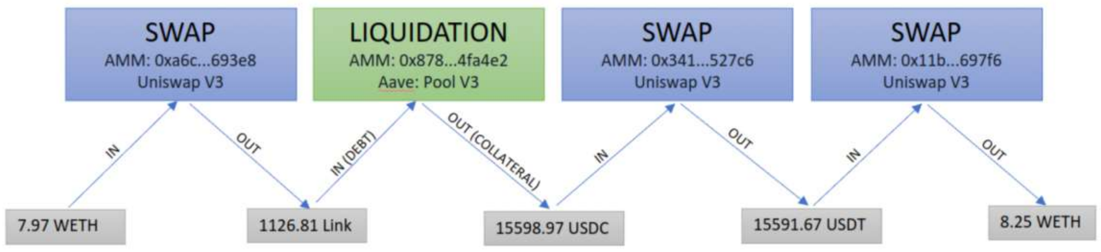


Figure 5.7: Visualization of the flow of tokens between the swaps and Liquidation of transaction 0x350...d640c.

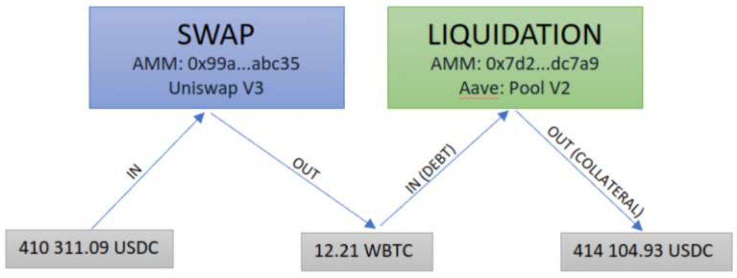
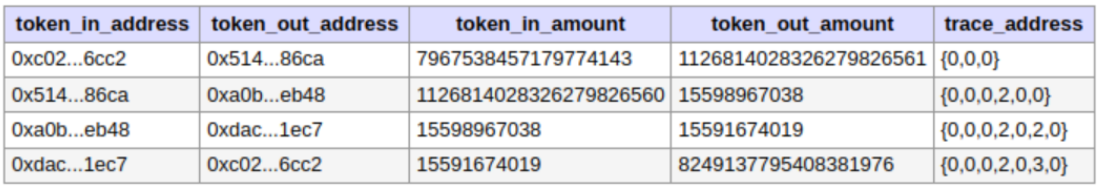


Figure 5.8: Visualization of the flow of tokens between the swap and Liquidation of transaction 0xb0d...0faa5.

The latter shows that this heuristic also allows a transaction to end with a Liquidation instead of a Swap.
If such a token flow can be constructed and the first "in" token is the same token as the last "out" token, then the Liquidation MEV value is calculated as the difference between the last "out" value and the first "in" value. Suppose such a token flow can not be constructed or the first "in" token is not the same as the last "out" token. In that case, the Liquidation MEV value is calculated as the difference between the collateral's and debt's values.
To implement this heuristic efficiently in MEV-Inspect, a view was created that represents this filtered token flow. This view combines and filters the swaps and Liquidation of a transaction and allows the ordering by the trace_address. Figure 5.9 shows the token flow in this view of the transaction 0x350...d640c.



token_in_address	token_out_address	token_in_amount	token_out_amount	trace_address
0xc02...6cc2	0x514...86ca	7967538457179774143	1126814028326279826561	{0,0,0}
0x514...86ca	0xa0b...eb48	1126814028326279826560	15598967038	{0,0,0,2,0,0}
0xa0b...eb48	0xdac...1ec7	15598967038	15591674019	{0,0,0,2,0,2,0}
0xdac...1ec7	0xc02...6cc2	15591674019	8249137795408381976	{0,0,0,2,0,3,0}

Figure 5.9: SQL view that combines swaps and Liquidations to represent the token flow of the transaction 0x350...d640c. The SQL query is documented in Listing 3

The full update of the Liquidation calculation in MEV-Inspect was implemented by overwriting the existing SQL query, which calculates the MEV value of Liquidations. This new query is listed in Listing 2.

Listing 2: Our suggested query for the MEV calculation of Liquidations, that implements our defined Heuristic

```sql
SELECT * FROM

SELECT
        NULL,
        received_token.token_address,
        debt_token.token_address,
        l.block_number,
        b.block_timestamp,
        l.protocol as protocol,
        l.transaction_hash,
        'liquidation' as type,
        (
          CASE
            WHEN first.token_in_address =
             ↪  last.token_out_address THEN
             ↪  last.token_out_amount
            ELSE l.received_amount
          END
        )
        *
        (
            SELECT usd_price
            FROM prices
            WHERE token_address = received_token.token_address
            AND timestamp <= b.block_timestamp
            ORDER BY timestamp DESC
            LIMIT 1
        )
        /POWER(10, received_token.decimals)

        -


        (
          CASE
            WHEN first.token_in_address =
             ↪  last.token_out_address THEN
             ↪  first.token_in_amount
            ELSE l.debt_purchase_amount
          END
        )
        *
        (
            SELECT usd_price
            FROM prices
            WHERE token_address = debt_token.token_address
            AND timestamp <= b.block_timestamp
            ORDER BY timestamp DESC
            LIMIT 1
        )
        /POWER(10, debt_token.decimals) as gross_profit_usd,
        (
            (
                ((mp.gas_used * mp.gas_price) +
                 ↪  mp.coinbase_transfer) /
                POWER(10, 18)
            ) *
            (
                SELECT usd_price
                FROM prices p
                WHERE
                    p.timestamp <= b.block_timestamp
                    AND p.token_address =
                     ↪  '0xeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee'
                ORDER BY p.timestamp DESC
                LIMIT 1
            )
        ) AS miner_payment_usd,
        mp.gas_used,
        mp.gas_price,
        mp.coinbase_transfer,
          mp.gas_price_with_coinbase_transfer,
          mp.miner_address,
          mp.base_fee_per_gas,
          ct.error as error,
          ARRAY[l.protocol]
      FROM liquidations l
      LEFT JOIN swaps_liquidations_merged first ON
       ↪  first.transaction_hash = l.transaction_hash AND
       ↪  first.trace_address =
      (
        SELECT first_inner.trace_address FROM
         ↪  swaps_liquidations_merged first_inner
        WHERE first_inner.transaction_hash = l.transaction_hash

        ORDER BY first_inner.trace_address ASC
        LIMIT 1
      )
      LEFT JOIN swaps_liquidations_merged last ON
       ↪  last.transaction_hash = l.transaction_hash AND
       ↪  last.trace_address =
      (
        SELECT last_inner.trace_address FROM
         ↪  swaps_liquidations_merged last_inner
        WHERE last_inner.transaction_hash = l.transaction_hash

        ORDER BY last_inner.trace_address DESC
        LIMIT 1
      )
      JOIN blocks b ON b.block_number = l.block_number
      JOIN tokens received_token ON
       (
        CASE
           WHEN first.token_in_address = last.token_out_address
            ↪  THEN received_token.token_address =
            ↪  last.token_out_address
           ELSE
received_token.token_address = l.received_token_address
         END
       )
      JOIN tokens debt_token ON
      (
        CASE
        WHEN first.token_in_address = last.token_out_address
         ↪  THEN debt_token.token_address =
         ↪  first.token_in_address
        ELSE debt_token.token_address = l.debt_token_address
      END
    )


    JOIN miner_payments mp ON
        mp.block_number = l.block_number AND
        mp.transaction_hash = l.transaction_hash
    JOIN classified_traces ct ON
        ct.block_number = l.block_number AND
        ct.transaction_hash = l.transaction_hash

    WHERE
        ct.trace_address = '{}' AND
        l.debt_purchase_amount > 0 AND
        l.received_amount > 0 AND
        l.debt_purchase_amount < 115792089237316195423570985008
        687907853269984665640564039457584007913129639935
```

Validation of the new Liquidation Query

To validate the new Liquidation calculation, we first reran the MEV calculation by inspection of the previously analyzed transactions (analyzed in section 5.1.1 and in section 5.1.1. The transaction 0xb0d...0faa5 now reports a profit of 3,757.79 $ and 0x350...d640c a profit of 18.53 $. These values represent exactly the manually calculated profit of these transactions in section 5.1.1 and in section 5.1.1.
In total, MEV-Inspect identified 426 Liquidations in the study's block range. Of these 426 Liquidations, 161 have a different profit calculated with the new Liquidation heuristic.

Figure 5.10 compares the result of the unchanged EigenPhi Liquidation calculation with the results from MEV-Inspect with the modified Heuristics and the results from the unmodified version of MEV-Inspect. It shows that the results of the modified MEV-Inspect heuristics are much closer to the results of the original EigenPhi results.

5.2.3 Conclusion and Discussion

This chapter compared two tools for detecting and quantifying MEV in transactions. The comparison revealed significant differences in the number of detected MEV transactions, with EigenPhi identifying ten times more transactions than MEV-Inspect. We then conducted case studies on specific MEV transactions, analyzing them trace log by trace

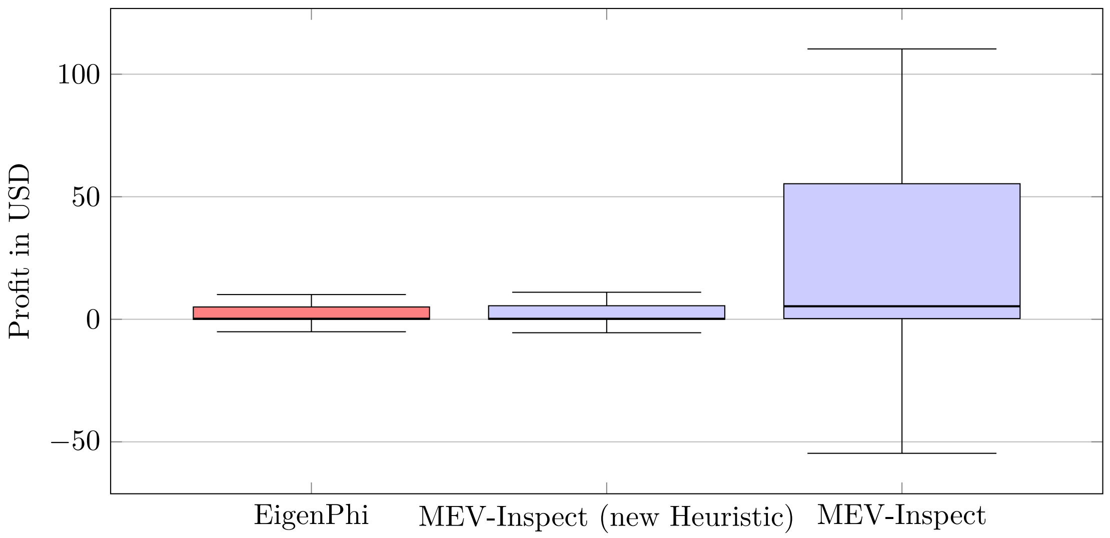


Figure 5.10: Comparison of Liquidation results, in the block range of 79801461 to 18580146. The first boxplot shows the results from EigenPhi, the second shows the results from MEV-Inspect after applying our new heuristic, and the third boxplot shows the original MEV-Inspect result.

log, and compared the results from these two tools with our own calculations. These case studies exposed several shortcomings in both MEV-Inspect and EigenPhi.

We identified a bug in the MEV detection logic based on these shortcomings and proposed a fix. Additionally, we introduced our own Liquidation heuristic, which considers the value gained from the difference between the bought debt and the acquired collateral, accounting for all swaps made before and after the Liquidation event.

This chapter reiterates some shortcomings previously mentioned in subsection 4.2.1. MEV-Inspect and EigenPhi are rule-based MEV detection tools, functioning on heuristics that identify transaction strategies. The differences highlighted in this chapter illustrate the challenges of rule-based MEV detection mechanisms, as their success relies on the quality of the heuristics. Unknown strategies remain undetected by design, necessitating continuous efforts to keep such tools up to date.

One problem that has accompanied MEV quantification is that since its first introduction by Daian et al. [DGK+20], no formal, clear, and exact definition of MEV has been established. While the community has proposed various definitions, this has only added to the confusion.

Initially, MEV stood for Miner Extractable Value, but it was later changed to Maximal Extractable Value to encompass not only PoW blockchains but also value extracted by actors other than miners. Another factor complicating MEV estimation is the influence of external factors, such as other cryptocurrencies, cross-chain interactions, and changes in existing cryptocurrencies.

Estimating MEV involves navigating a complex, probabilistic, and dynamic environment. Certain actors' behavior, resources, and capabilities can only be estimated [JSSW21]. For example, the pseudonymous nature of cryptocurrencies makes it difficult to determine all the assets or addresses of a specific MEV actor. When analyzing an MEV extracting transaction, it is possible that the sender's address does not profit directly but rather another address linked to the searcher. The MEV searcher could even profit indirectly on another blockchain by utilizing bridges, i.e. extracting cross-chain MEV. This complexity makes it especially challenging to estimate an upper bound for MEV.

The value of MEV has been made more explicit by introducing Proposer-builder separation (PBS) in Ethereum. Although PBS is not included in the Ethereum consensus protocol, a market has emerged where block builders can purchase block space from block proposers. As of July 27, 2024, 89.97 % [62] of blocks have been proposed through a PBS mechanism. Block builders bid against each other to purchase this block space. These bids can be seen as a lower bound of MEV, as it is implied that an economically rational actor would aim to pay less for the block space than it can gain from MEV.

If MEV searching is relatively decentralized and MEV searchers generally employ similar strategies, they will compete against each other, driving up bids to match the block's MEV value. In this scenario, the MEV Boost value would closely approximate the actual MEV.

[62]https://mevboost.pics/, In a timeframe of 14 days

CHAPTER 6

Multi-Block MEV

In this chapter, we will explore Multi-block MEV. First, we will introduce two strategies an MEV searcher can use to build a consecutive sequence of blocks: controlling a sizable percentage of the overall stake, e.g., operators of staking pools or other large entities such as exchanges that perform staking on behalf of their clients and using MEV-Boost. We will then analyze these strategies to identify any indications of their current use. Following this, we will propose a straightforward Multi-block strategy and evaluate the Ethereum blockchain to assess its economic viability. We will begin by explaining our data collection process.

6.1 Data retrieval

To conduct the analyses in this chapter, we required the following data:

- **Block data** Block Number, Slot Number, Block Proposer
- **Validator data** Public Key, Staking Pool Association
- **MEV data** MEV per transaction
- **MEV-Boost data** Proposer Public Key, Builder Public Key, Value paid to Proposer
- **Slot data** Associated Block, missed / not missed

An in-depth analysis of the entire Ethereum blockchain for MMEV requires significant resources and is beyond the scope of this thesis. We therefore focused on blocks 17,980,146

[1] to 18,580,146 [2], corresponding to slot numbers 7,166,892 to 7,771,817, which comprise 600,000 slots.

We decided to store all data in a PostgreSQL database because of our familiarity with PostgreSQL. Initially, we hosted the database on an AWS instance [3]. However, as the data grew, we moved it to a server the *Security and Privacy Group* at the University of Vienna provided to reduce costs. The database was run on the server using Docker and Docker Compose, along with an instance of Adminer [4], which provides a web GUI to execute SQL queries and browse the data.

The first step was to fetch the block data. We began by writing a JavaScript program connected to a Lighthouse client [5], hosted at the University of Vienna. The Lighthouse version on this server was "Lighthouse/v4.2.0-c547a11/x86_64-linux." [6]. This JavaScript program looped through all slots within the study timeframe and queried the Lighthouse API [7] to obtain the block information for each slot.

If the API returned a 404 status code for a specific slot, we stored this slot in the missed_slot table; otherwise, we stored the block information in the block table. Due to the many slots, these calls were parallelized using Node.js Worker Threads [8]. Out of 600,000 blocks, we identified 4,925 missed slots, which accounts for 0.82 % of all slots.

We further needed the information on whether a validator was part of a staking pool. This alliance with a staking pool is not recorded on the Ethereum blockchain. However, it is publicly known which proposer proposed each block. A proposer's public key becomes public when they deposit 32 ETH into the deposit contract. Once a validator is added to the set of validators, a validator_index is used to reference a *ValidatorRecord* containing the public key [9]. The Beacon API's/eth/v1/beacon/states/state_id/validators endpoint [10] allows you to query this validator's public key and index. We used this Beacon API endpoint to extract the proposer's public key.

Anton Wahrstätter published a dataset [11] containing mappings between proposers' public keys and their pool alliances. We imported this dataset into our local PostgreSQL database and added it to our validator information by mapping it using the public key. Using the proposer_index in the block header, we labeled each block with the Staking Pool information.

[1]https://etherscan.io/block/17980146
[2]https://etherscan.io/block/18580146
[3]https://aws.amazon.com/rds/postgresql/ Last access 2024-08-01
[4]https://hub.docker.com/_/adminer/ Accessed on 2024-08-01
[5]A popular Beaconchain client, for more details, see subsection 2.2.7
[6]Queried at 2023-12-01
[7]Endpoint eth/v2/beacon/blocks/<slot>, Reference: https://ethereum.github.io/beacon-APIs/#/Beacon/getBlockV2
[8]https://nodejs.org/api/worker_threads.html
[9]https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/validator.md Accessed on 2024-03-09
[10]https://ethereum.github.io/beacon-APIs/#/Beacon/getStateValidators Accessed on 2024-03-09
[11]https://mevboost.pics/data.html Accessed on 2024-07-31

To gather information on MEV-Boost, we used the published MEV-Boost data from Anton Wahrstätter [12]. This dataset contains block rewards recorded by several MEV-Boost relays and the public keys of validators and builders for all blocks.

The data was provided as a Parquet file and imported using PyArrow [13]. We utilized the Python data analysis library Pandas [14] to filter Wahrstätter's data to match our test block range. The dataset covers blocks from 17,980,146 to 18,373,887, encompassing 393,230 blocks. Since this was a raw dataset, various validation and cleaning steps were necessary.

First, we identified 1,133 blocks in our database's MEV-Boost data recorded as missed slots. These blocks did not make it into the finalized blockchain due to forking, validator inactivity, or late publication. We could delete these entries since we had a finalized view of this data range and knew these were missed slots.

Further validation revealed multiple rows for each block number, primarily because different relays recorded the same block in 918,676 cases. The data came from multiple relays [15], and we assume builders submit their blocks to multiple relays to increase their chances of inclusion in the finalized chain, resulting in multiple entries. We removed these duplicates, leaving 362 blocks for the same slot with different mevboost_value entries, which indicate how much the block builder pays for block space. Some rows also had different proposer public keys, likely because each relay reported one "winning" block for its auction. However, only one slot per block can be published, so we focused on identifying rows that ended up in the finalized blockchain.

We used the previously retrieved public validator key to eliminate rows containing a proposer public key different from the one that ended in the finalized blockchain.

The final step in data cleaning was merging blocks with the same builder. Some rows had identical data except for a different builder public key. We used a key label index by Anton Wahrstätter to label these builders and remove duplicates.

After cleaning, the dataset contains 393,230 rows, with one entry per block that is not missed. We now have a lower bound of the number of blocks proposed using MEV-Boost, the validator and builder public keys, and the proposer and builder's staking pool affiliations.
The next data needed are the MEV values for all transactions, which we obtained in section 6.1.

[12]https://mevboost.pics/data.html Accessed on 2024-03-22
[13]https://arrow.apache.org/docs/python/index.html Accessed on 2024-07-30
[14]https://pandas.pydata.org/ Accessed on 2024-07-30
[15]https://github.comnerolationmevboost.picsblobmainscriptsparse_data_api.py Accessed on 2024-03-22

6.2 Building a Multi-Block sequence

This section will discuss how an MEV searcher can build a Multi-block sequence. We will first discuss two potential forms to secure a consecutive sequence of blocks through the Proof of Stake protocol and using MEV-Boost. After that, we will analyze these two strategies in a custom date range to find evidence of Multiple block MEV.

6.2.1 Consecutive blocks in Ethereum through Proof of Stake protocol

In Proof of Work Ethereum, the miner who solved the puzzle was unknown beforehand. The selection depended on the miner's proportional hash power and luck. In Proof of Stake Ethereum, the next proposer is chosen using a pseudo-random algorithm called RANDAO. This mechanism selects all proposers for an epoch t at the end of epoch $t-2$ [16], making the entities who propose in an epoch publicly known.

As of July 30, 2024, Ethereum has 1,050,134 validators [17]. This large number of validators makes the probability of selection minimal and the likelihood of being selected multiple times in an epoch or for consecutive slots even smaller. However, validators often collaborate in pools, allowing entities to participate as validators even without the required 32 Ether. All participants' stakes are combined and managed as a single entity in these pools [18]. While there are potential centralization concerns, staking pools control more deposited stakes and, therefore, more slots than individual stakes. For example, as of July 30, 2024, the staking pool Lido controls 8.84 % of validators in Ethereum [19]. This increases the likelihood of a pool being selected to propose blocks and, consequently, the possibility that a pool can propose multiple blocks in sequence.
To calculate the probability of k consecutive events in a trial of $n = 32$ (one epoch), for a pool with a share of p, Alvaro Revuelta [Rev22] proposes the following approximation:

$$q_n \approx \frac{1 - px}{(k + 1 - kx)(1 - p)} \frac{1}{x^{n+1}}$$

where x is the closest real root to the result of $1 - x + (1 - p)p^k x^{k+1} = 0$.

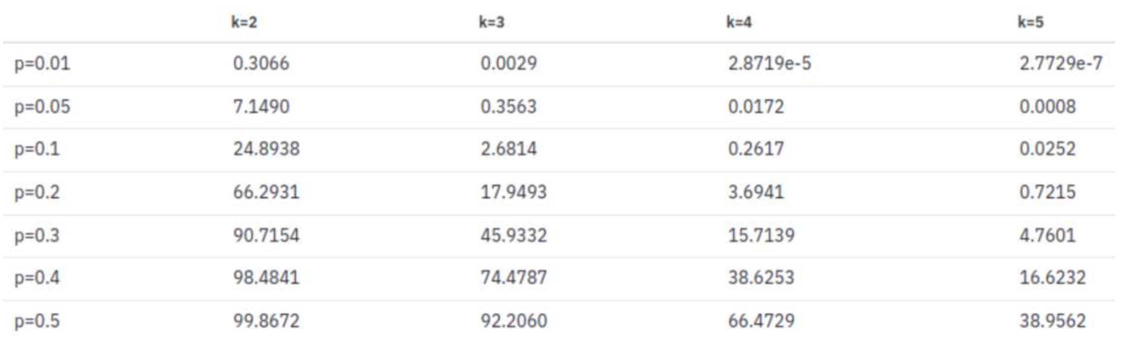
Figure 6.1 shows the calculated probabilities using Revuelta's formula for a pool controlling a share of p validators to obtain k consecutive blocks in an epoch. This estimation suggests that if a staking pool controls 10 % of all validators, it has a probability of controlling two consecutive blocks of nearly 25 %. This probability applies to every epoch, which occurs approximately every 6.4 minutes. Thus, it demonstrates that a sufficiently sizeable staking pool can control consecutive blocks solely due to its significant share of validators.

[16] https://ethresear.ch/t/selfish-mixing-and-randao-manipulation/16081

[17] https://beaconcha.in/

[18] There are other pool designs where the pool does not act as a single entity, but for simplicity, we consider staking pools as single entities, i.e., pool operators control how all participating individuals act.

[19] https://beaconcha.in/pools#distribution



	k=2	k=3	k=4	k=5
p=0.01	0.3066	0.0029	2.8719e-5	2.7729e-7
p=0.05	7.1490	0.3563	0.0172	0.0008
p=0.1	24.8938	2.6814	0.2617	0.0252
p=0.2	66.2931	17.9493	3.6941	0.7215
p=0.3	90.7154	45.9332	15.7139	4.7601
p=0.4	98.4841	74.4787	38.6253	16.6232
p=0.5	99.8672	92.2060	66.4729	38.9562

Figure 6.1: Probabilities of staking pools gaining a consecutive sequence of blocks. p denotes the relative size of a staking pool compared to all validators and k the length of a consecutive sequence of blocks. Figure from [Rev22]

6.2.2 Consecutive blocks through MEV-Boost

The Ethereum Foundation plans to include Proposer-builder separation (PBS) in the Ethereum protocol. Meanwhile, MEV-Boost by Flashbots has established itself as a widespread PBS implementation that is not part of the official Ethereum protocol. As of July 27, 2024, according to Anton Wahrstätter, MEV-Boost is widely accepted, with 89.97 % [20] of all blocks being proposed using this mechanism.

MEV-Boost creates a marketplace for block space by introducing a relay between block proposers (validators) and block builders. It is an optional extra client that validators can run. This client communicates with the consensus client and forwards blocks to it for publication. Block builders aggregate bundles into profitable blocks and send them to the MEV-Boost relay. These relays verify the blocks and forward only a block header, without any transactions, back to the MEV-Boost client. The transactions are withheld to prevent block validators from stealing and publishing the transaction bundles for their own gain.

The header indicates how much the block builder will pay for the block space. The validator can then select the highest-paying block, sign it, and return it to the relay. Once the relay receives the signed header, it returns the payload to the MEV-Boost client and the validator, who then publishes the block.
This sealed-bid auction marketplace allows builders without sufficient stake to purchase block space by making competitive bids. As discussed in subsection 6.2.1, it is extremely unlikely to publish multiple blocks in a row for an individual validator. [Fla24]

PBS through MEV-Boost enables a block searcher to purchase multiple blocks. One can secure a sequence of blocks by offering the highest price for consecutive blocks. However, unlike staking pools, which have guaranteed multiple consecutive blocks (as explained in subsection 6.2.1), the MEV searcher does not have a guaranteed sequence of blocks using

[20]https://mevboost.pics/, In a timeframe of 14 days

MEV-Boost. The MEV-Boost auction runs in the slot before the target slot. To secure a consecutive sequence of k slots, the searcher must outbid all other searchers k times in a row. Because strategies running over multiple blocks are not atomic, the searcher risks only partially executing their strategy and potentially not profiting or even incurring losses.

6.2.3 Analysis

In the following section, we will analyze the two methods for acquiring a consecutive sequence of blocks using the data gathered and described in section 6.1. We aim to determine whether blocks within a sequence contain more MEV than others.
We will use MEV data to assess whether consecutive sequences assigned to staking pools have an increase in MEV value. For sequences built through MEV-Boost, we will use the MEV-Boost value, which is the amount paid to the block proposer, as a metric.

Consecutive Block sequence through PoS Ethereum

In this analysis, we aim to determine whether the MEV (Miner Extractable Value) in block sequences is statistically different from that in other blocks. Specifically, we focus on blocks not proposed through MEV-Boost. We define a block sequence as a series of blocks proposed by block proposers within the same pool, assuming that the pool implements strategies across its various validators.
Details on how we identified which blocks were published via MEV-Boost are provided in section 6.1. The study examines blocks numbered between 17980146 and 18373887, during which we have an estimate of which blocks were published using MEV-Boost. Within this period, 29,912 blocks were not published through MEV-Boost, accounting for approximately 7.6 % of all blocks in the study timeframe.
Using the query in Listing 1, we identified all block sequences, with the results visualized in Figure 6.2. We discovered 92 unique block sequences, most of which were of length 2, with one sequence extending to length 7.

We retrieved MEV data using EigenPhi [Eig24a] to assess whether the MEV value differs significantly between blocks within a sequence.
We conducted a Mann-Whitney U test to determine whether there is a statistically significant difference in the total MEV value between these blocks. The null hypothesis H_0 posits that both samples originate from the same distribution, while the alternative hypothesis H_1 suggests they do not. We set the threshold for statistical significance at $\alpha = 5\%$.

The Mann-Whitney U test was implemented in Python, utilizing psycopg2 [21] to fetch data from our database and the SciPy [22] package to conduct the test.

[21]https://pypi.org/project/psycopg2/ Accessed on 2024-07-31
[22]https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mannwhitneyu.html Accessed on 2024-07-31

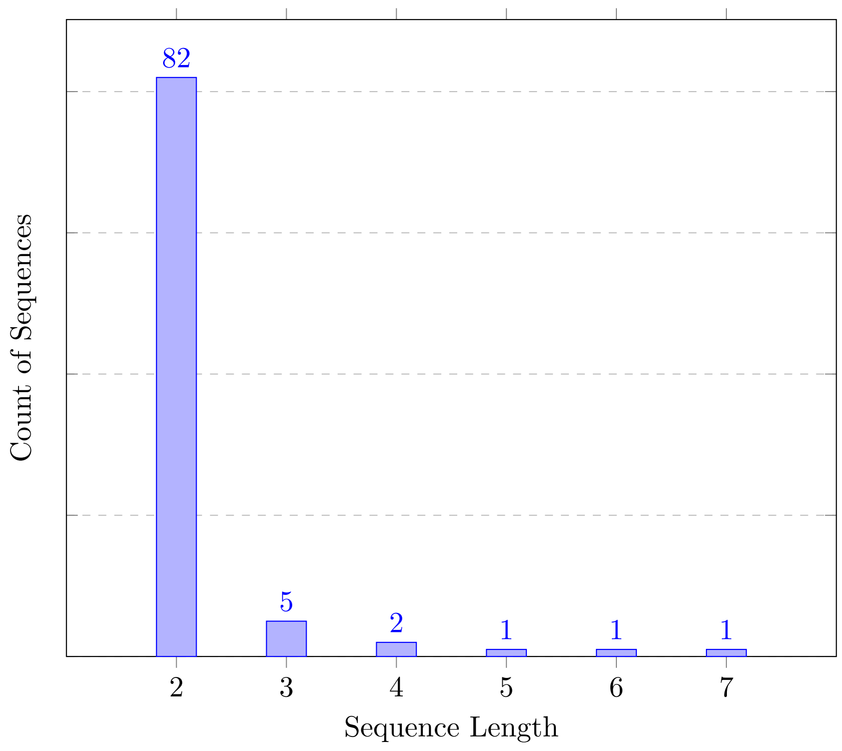


Figure 6.2: Occurrences of consecutive block sequences proposed by validators belonging to the same staking pool, in the block range of 79801461 to 1858014

The test yielded a p-value of 0.003, allowing us to reject the null hypothesis H_0. We conclude that the samples are from different distributions, indicating that the MEV value in consecutive blocks significantly differs from that in other blocks.
However, as shown in Figure 6.3, blocks in consecutive sequences generally exhibit lower MEV profits than non-MEV-Boost blocks. It is important to note that, to our knowledge, all existing MEV detection mechanisms focus on Single-block MEV and do not detect strategies deployed over consecutive blocks. Consequently, we cannot conclude that the MEV gained by staking pools in multi-block sequences is lower than in other blocks, as Multi-block MEV attacks were potentially undetected.

Consecutive Block sequence through MEV-Boost

In this analysis, we explore the second method of acquiring a consecutive sequence of blocks by purchasing block space through MEV-Boost. First, we investigate the frequency of sequences originating from the same builder within our study range. Then we analyze the pricing of this block space to determine whether block builders pay more for blocks in a sequence than for individual blocks.

Using the data described in section 6.1, the next step is identifying consecutive blocks from the same builder and analyzing changes in the *mevboost_value*. The *mevboost_value*

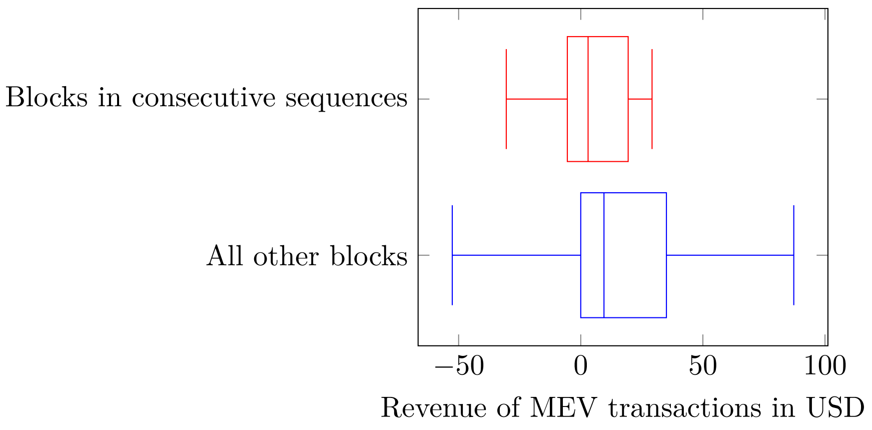


Figure 6.3: A comparison of the MEV revenue reported by EigenPhi of blocks in consecutive sequences (red) vs. other blocks (blue) for blocks that were published without MEV-Boost, in the block range of 79801461 to 1858014

is the price a block builder pays to a block producer for the block space. Our analysis shows that consecutive sequences are relatively common, as depicted in Figure 6.4. We identified 47,808 sequences of at least two blocks from the same builder, indicating that about 12 % of all blocks in our study timeframe were part of a Multi-block sequence. Additionally, we detected some longer sequences, including one sequence of twelve blocks and a total of 26 sequences with more than nine blocks.

We aim to investigate whether builders pay a premium for an entire sequence of blocks, not just for individual blocks within a sequence. To do this, we calculate the total value paid by the block builder for the entire sequence (query listed in Listing 2. We then compare these accumulated values to the median MEV-Boost value multiplied by the sequence length. We assume that builders who do not employ a Multi-block strategy are unlikely to pay a premium for consecutive block sequences and would aim to pay around the median price for the entire sequence.

The results of this analysis are visualized in Figure 6.5. The red line represents the median price of a block multiplied by the number of blocks in the sequence, while the blue line shows the actual payments made for consecutive blocks. This figure illustrates that builders pay more than the expected median prices for a series of consecutive blocks compared to individual blocks. These findings appear to confirm the conclusions of Jensen et al. [JvWR23] and suggest that MEV searchers utilize Multi-block strategies.

6.3 Missed slot strategy

In this section, we formulate and analyze the so-called "Missed Slot" strategy. This straightforward strategy involves a block searcher intentionally missing a slot to capitalize on the accrued MEV. The searcher must have secured the right to propose two consecutive blocks for this strategy to be effective. The methods by which this can be achieved are

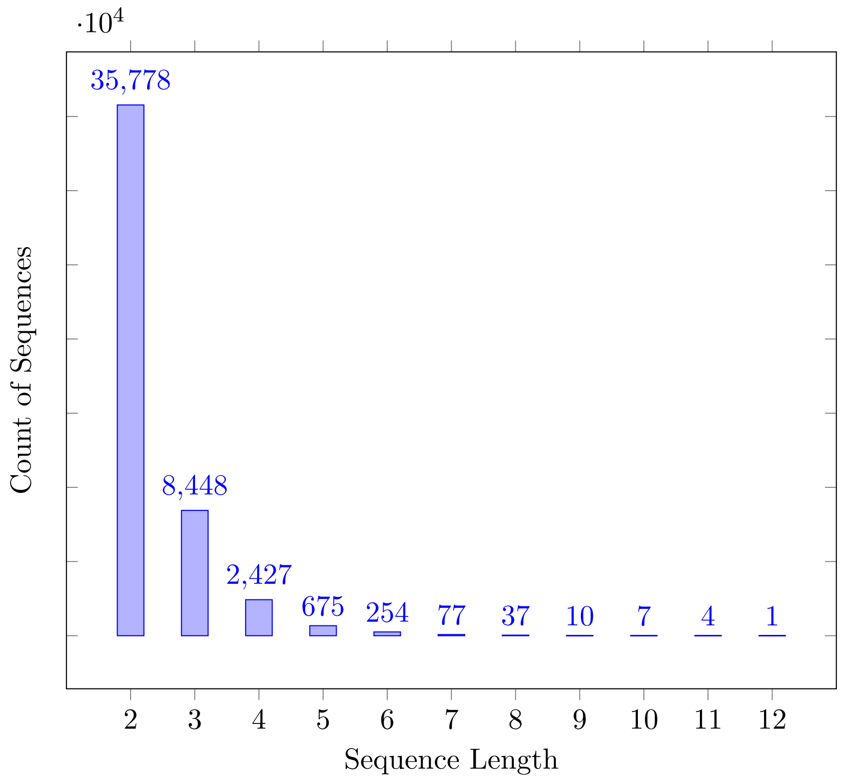


Figure 6.4: Occurrences of consecutive block sequences with the same builder, in the block range of 79801461 to 1858014

discussed in section 6.2. We utilize the retrieved data to analyze this strategy using the methods described in section 6.1.

6.3.1 Evaluation

In 600,000 slots or 595,075 blocks, MEV-Inspect identified MEV transactions in 62,291 blocks. Within these blocks, it detected 110,084 Arbitrage trades, 327,680 Liquidations, 83,307,529 payment transactions to proposers, 590,144 Sandwich trades, and 20,015,650 swap trades.
To address the first part of RQ_2, which asks whether the value of MEV in the block following a missed slot is higher, we first analyze the total MEV value of blocks that do not follow a missed slot, and those that occur immediately after or two slots after a missed slot. Because of the large number of blocks without detected MEV transactions, the median MEV value of these blocks is 0. However, when considering only blocks with detected MEV transactions, the median value is 8.28 \$. Blocks immediately after a missed slot have a median MEV value of 10.69 \$. The blocks two slots after a missed slot have a median MEV value of 9.42 \$. An overview of these values can be seen in Table 6.1 and in the box plots in Figure 6.6.

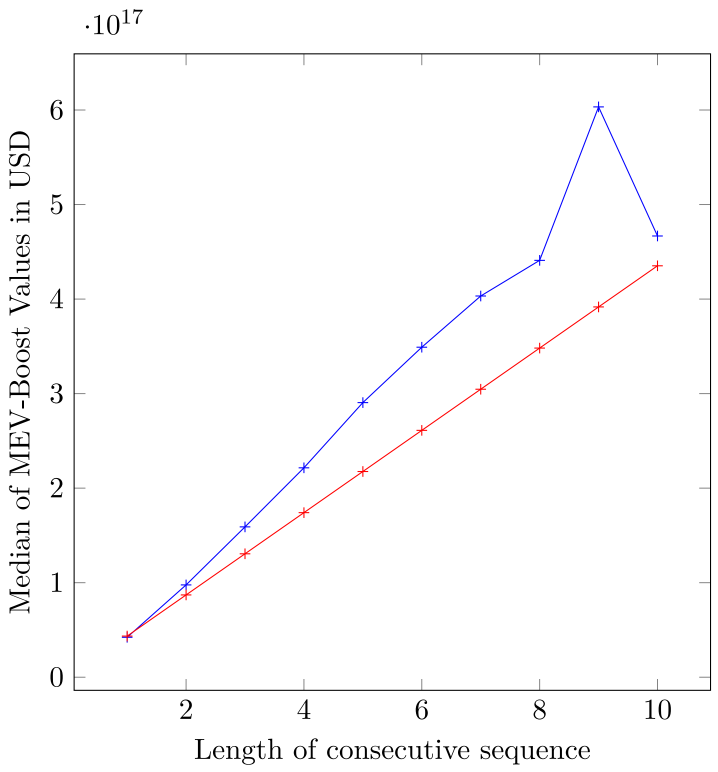


Figure 6.5: The red line represents the median block value multiplied by x, where x is the number of consecutive blocks. A rational builder is expected to pay around this value for consecutive blocks if they are not specifically targeting them. The blue line represents the actual amount builders pay for consecutive blocks. At $x = 1$, it shows the median value of all individual blocks. It shows the median total price for a consecutive series of two blocks at $x = 2$, $x = 3$ for a series of three blocks, and so on.

	Min	1st Quantile	Median	3rd Quantile	Max
+1 after missed slot	$10^{-3.068}$$	4.98 $	10.69 $	24.72 $	10,327.51 $
+2 after missed slot	0.17 $	4.57 $	9.39 $	24.12 $	3,294.00 $
All other blocks	$10^{-14.613}$$	4.28 $	8.28 $	21.02 $	587,876.00 $

Table 6.1: Overview of MEV block values

A Mann-Whitney U test was performed to evaluate the statistical significance of the difference in the MEV value between regular blocks and those following a missed slot. The null hypothesis H_0 states that both samples come from the same distribution, while the alternative hypothesis H_1 suggests that the two samples originate from different distributions. Statistical significance is defined by a significance level of $\alpha = 5\%$.

The Mann-Whitney U test results, comparing datasets of all blocks immediately after a missed slot with all other blocks, yielded a p-value of 0.001. Therefore, we can

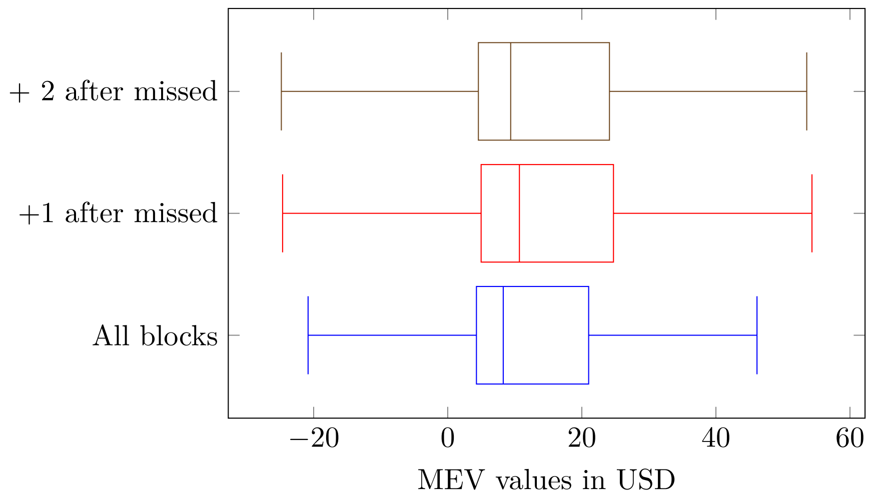


Figure 6.6: MEV values in all blocks, one slot after a missed slot and two slots after a missed slots, in the block range of 79801461 to 1858014

reject the null hypothesis H_0, indicating that the differences are statistically significant. However, when comparing blocks two slots after a missed slot with all other blocks, the p-value is 0.116, which means the null hypothesis H_1 cannot be rejected.

The results of this analysis suggest that a higher MEV value can be expected immediately after a missed slot. Comparing the medians, an MEV searcher can anticipate a 15-30 % higher yield if they propose immediately after a missed block. However, a statistically significant gain was not observed for two slots after a missed slot.
The median price for a block through MEV-Boost during the study timeframe was 40,227,794,367,156,850 Wei or approximately 0.0402 Ether.
With a median exchange rate of 1,643.70 $ during this period, a searcher paid roughly 66 $ for block space. Therefore, the increase in MEV from intentionally missing a slot does not constitute an economically viable strategy.

6.4 Conclusion

In this chapter, we introduced two strategies that a block searcher could use to obtain a consecutive sequence of blocks. The first strategy involves calculating the probability of securing two consecutive slots. The second strategy involves purchasing block space through the blinded auction platform MEV-Inspect.
Our data shows that the same staking pool sometimes obtains consecutive sequences, and the MEV in these sequences is statistically different from that in other blocks. However, since MEV detection typically focuses on Single-block MEV strategies, this analysis alone does not prove that MEV searchers apply multiple-block strategies.
We also observed that block builders paid a premium for blocks in consecutive sequences. Assuming economically rational behavior, this suggests that Multi-block strategies are

being employed using MEV-Boost to secure such sequences.
Lastly, we analyzed the "Missed Slot" strategy, where a builder intentionally misses a block to increase MEV in the subsequent block. Although we found that blocks immediately following a missed slot had a statistically significant increase in MEV value, the increase was insufficient to make this strategy economically viable.

CHAPTER 7
Conclusion

This work focused on Maximal Extractable Value. We laid the foundation by covering the technical aspects of Ethereum and introduced the problem and various forms of MEV in Ethereum. A systematic literature review analyzed current state-of-the-art MEV detection techniques, which we summarized and categorized. We investigated some shortcomings of rule-based MEV detection techniques and introduced a new Heuristic for quantifying the profit made by Liquidation transactions. Finally, we empirically investigated the topic of Multi-Block MEV.

This chapter will revisit the initially proposed research questions and highlight how we addressed them and their results. This chapter will also discuss potential Future Work on this topic.

7.1 Results

> *RQ_1: What is the current state-of-the-art in MEV attacks, and what mitigation strategies have been implemented or proposed?*

This research question was addressed by categorizing the currently observed MEV attacks, which include Frontrunning, Backrunning, Sandwich attacks, Arbitrage, Transaction Replay, and Clogging. We also explored additional dimensions of MEV, such as cross-domain MEV and Multi-block MEV, which are not confined to Single-block targeted attacks. Beyond the direct losses suffered by victims, we highlighted MEV's negative externalities, such as network congestion, reduced transaction throughput, and increased gas prices. Furthermore, the high MEV value may encourage a new category of attacks on the blockchain consensus, where attackers attempt to rewrite blocks to exploit past MEV opportunities. We also introduced mitigation strategies to prevent MEV, such as achieving order fairness and making MEV more transparent and accessible to reduce centralization risks.

> *RQ_2: What are the state-of-the-art methods for detecting and quantifying Single-block Maximal Extractable Value in Ethereum, and what are the limitations of these methods?*

To address this research question, we conducted a systematic literature review. From an initial pool of 1,004 papers and tools, we selected 16 relevant works to analyze their MEV detection and quantification methods. We classified these works into various categories, with a significant distinction in detection strategies: some employed rule-based heuristics to identify MEV transactions, while others utilized machine learning techniques.

Rule-based approaches have the advantage of having easily explained strategies and detection heuristics, making their results reproducible. However, these methods require precise knowledge of the mechanisms they detect and must be updated with each newly published technique. Additionally, they necessitate a thorough understanding of Application Binary Interfaces (ABIs), which must be revised for each new DeFi service and variations of existing services that emerge. This reliance can lead to a high potential for false negatives, as some strategies may go undetected, particularly when there are small variations in tactics.

Conversely, machine learning approaches are not dependent on specific ABIs, reducing the maintenance effort. However, they require a high-quality dataset, typically generated using rule-based mechanisms. This reliance means that a machine learning-based method is only as effective as the underlying dataset and is unlikely to detect new forms of MEV that were not present in the training data.

In our classification of various MEV tools, we observed differing results from two rule-based MEV detection tools, MEV-Inspect and EigenPhi. Based on case studies, a direct comparison of these tools revealed some shortcomings. As a result of this analysis, we proposed specific improvements to MEV-Inspect, including a new Liquidation heuristic. These analyses underscored some of the limitations of rule-based MEV detection mechanisms.

MEV detection is inherently challenging due to several factors: the lack of a clear definition of MEV within the scientific community, the incentives for MEV searchers to conceal their strategies, and the pseudonymous nature of blockchains, which obscures the profiteering of some MEV strategies. We recommend using the MEV-Boost value as a lower bound for MEV quantification. An economically rational MEV searcher would aim to pay less for block space than the extracted MEV. If MEV searching is relatively decentralized and searchers generally employ similar strategies, they will compete, driving up bids to reflect the block's MEV value. In this case, the MEV-Boost value would closely approximate the actual MEV.

> *RQ$_3$: How can an MEV searcher gain control over consecutive sequences of blocks, and are there any indications that these methods are currently being employed?*

To address the third research question, we examined two potential methods for an attacker to acquire a Multi-block sequence. The first method involves leveraging chance in Ethereum's Proof of Stake system, where the attacker needs to hold a significant stake in Ethereum. Realistically, this can only be achieved by participating in a staking pool. We calculated the necessary share of the total stake required for an attacker to obtain a block sequence naturally. Our analysis of on-chain data demonstrated that such sequences are not uncommon and that their frequency aligns with our theoretical calculations. We also found that the MEV in these block sequences is statistically distinct from that of other blocks. However, this finding is limited because our MEV calculations per block relied on rule-based detection mechanisms that did not incorporate heuristics for MMEV attacks.

The second method for obtaining a consecutive block sequence involves purchasing blocks through a Proposer-builder separation (PBS) marketplace, the most prevalent of which is MEV-Boost. Our analysis of published MEV-Boost data revealed that MEV searchers do, in fact, purchase consecutive sequences and, on average, pay a premium over the typical block price. This behavior indicates that MEV searchers are exploiting and benefiting from MMEV opportunities.

> *RQ$_4$: Is the value of MEV in the block following a missed slot statistically higher, and would this make an attack where the attacker intentionally misses a block economically viable?*

We utilized a Beacon node to identify missed slots to address the final research question. We employed the open-source tool MEV-Inspect to analyze the MEV value of blocks immediately following these missed slots and blocks two slots after a missed slot. Our analysis revealed that the MEV value of blocks immediately following a missed slot was statistically significantly higher than that of all other slots. However, no statistical difference was observed for blocks two slots after a missed slot.

Blocks following missed slots exhibited an MEV increase of 15-30 %. Despite this increase, the findings do not suggest that intentionally missing a slot would be economically viable.

7.2 Future Work

MEV remains a crucial topic for research due to its significant financial impact and negative externalities. This thesis has classified current MEV detection tools, focusing on the primary categorization of rule-based versus machine-learning-based approaches. Future research could compare these detection methods to determine which approach offers better recall value.

Additionally, we propose to use the MEV value provided by MEV-Boost and similar applications as a baseline for MEV detection. Future analyses could compare the results of various detection tools with the MEV-Boost value, which serves as a lower bound for MEV quantification.

In this work, we analyzed Multi-block MEV (MMEV) and identified indications that some participants employed MMEV strategies. Further research could focus on understanding these specific strategies. Potential areas of exploration include Time Weighted Average Price (TWAP) oracles, governance contracts, and Automated Market Makers (AMMs). TWAP oracles provide price data over multiple blocks, and a Multi-block attack that censors only buy or sell orders could manipulate the reported price. Governance contracts allow stakeholders to vote on proposals, execute changes, and manage the decentralized governance of protocols and applications. They often have a block delay between determining voter eligibility and the actual vote, and controlling a sequence of blocks could exploit this governance mechanism. Willetts et al. [WH24] introduced a Multi-block MEV attack on dynamic AMMs and demonstrated its feasibility in simulations. However, to our knowledge, this attack has not been observed on-chain, making it a potential subject for further analysis.

Another research topic could be quantifying and detecting MMEV attacks, similar to the single-block MEV detection mechanism introduced in this thesis.

Overview of Generative AI Tools Used

Grammarly
The Grammarly "Premium" Version was used as a browser extension to refine the grammar and fix spelling mistakes.
Chat GPT (4o)
Chat GPT (4o) was used for rephrasing and rewording sentences. My written text was copied along with the prompt: "The following text is a section from my master thesis; find and fix mistakes and rephrase the sentences to increase the clarity and readability of the text. List all the changes that were done.". The result was then used to improve the phrasing and fix any mistakes.
Chat GPT was also used to generate first drafts of LaTeX tables and figures.

List of Figures

List of Tables

Bibliography

[AFG$^+$15] Edoardo Aromataris, Ritin Fernandez, Christina M Godfrey, Cheryl Holly, Hanan Khalil, and Patraporn Tungpunkom. Summarizing systematic reviews: methodological development, conduct and reporting of an umbrella review approach. *JBI Evidence Implementation*, 13(3):132–140, 2015.

[B$^+$14] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1, 2014.

[BAH$^+$23] Bert, Apriori, Hasu, Nfactorial, Masterdai, Kz, Josojo, Nikete, 0xEvan, boz1, and et al. Mev-share: Programmably private orderflow to share mev with users, Feb 2023.

[BG17] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[Blo70] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[CDG$^+$24] Yash Chaurasia, Parth Desai, Sujit Gujar, et al. Mev ecosystem evolution from ethereum 1.0. *arXiv preprint arXiv:2406.13585*, 2024.

[CH23] Facundo Carrillo and Elaine Hu. Mev in fixed gas price blockchains: Terra classic as a case of study. *arXiv preprint arXiv:2303.04242*, 2023.

[CHHW24] Tianyang Chi, Ningyu He, Xiaohui Hu, and Haoyu Wang. Remeasuring the Arbitrage and Sandwich Attacks of Maximal Extractable Value in Ethereum. Technical report, May 2024. arXiv:2405.17944 [cs] type: article.

[CKWN16] Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 154–167, 2016.

[DC$^+$85] Nicolas De Condorcet et al. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix.* Cambridge University Press (Digital Version), 1785.

[DE20] Yael Doweck and Ittay Eyal. Multi-party timed commitments. *arXiv preprint arXiv:2005.04883*, 2020.

[DGK+20] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.

[Dou02] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[Eig24a] EigenPhi. Eigenphi. https://eigenphi.io/, 2024.

[Eig24b] EigenPhi. How do we calculate the profit and loss of an mev transaction? https://eigenphi.substack.com/p/calculate-profit-and-cost-of-mev, Jan 2024.

[EMC20] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: front-running attacks on blockchain. In *Financial Cryptography and Data Security: FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*, pages 170–189. Springer, 2020.

[Eth21] EthTx. Ethtx ethereum transaction decoder (community version). https://github.com/EthTx/ethtx_ce, 2021.

[Eth24a] Ethereum.org. Accounts. https://ethereum.org/en/developers/docs/accounts/, Jun 2024.

[Eth24b] Ethereum.org. Gas and fees. https://ethereum.org/en/developers/docs/gas, Mar 2024.

[Fla] Ltd Flashbots. Flashbots docs. https://docs.flashbots.net/flashbots-auction/overview.

[Fla24] Ltd Flashbots. Flashbots docs. https://docs.flashbots.net/flashbots-mev-boost/architecture-overview/block-proposal, Jun 2024.

[GKL24] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. *Journal of the ACM*, 71(4):1–49, 2024.

[Han22] Magnus Hansson. Arbitrage in crypto markets: An analysis of primary ethereum blockchain data. *Available at SSRN 4278272*, 2022.

[HG22] Hasu and Stephane Gosselin. Why run mev-boost?: Flashbots, Jun 2022.

[HPPM22] Neal R. Haddaway, Matthew J. Page, Chris C. Pritchard, and Luke A. McGuinness. Prisma2020: An r package and shiny app for producing prisma 2020-compliant flow diagrams, with interactivity for optimised digital transparency and open synthesis. *Campbell Systematic Reviews*, 18(2):e1230, Jun 2022.

[HPS24] Lioba Heimbach, Vabuk Pahari, and Eric Schertenleib. Non-Atomic Arbitrage in Decentralized Finance. Technical report, April 2024. arXiv:2401.01622 [cs, q-fin] type: article.

[HW22] Lioba Heimbach and Roger Wattenhofer. Sok: Preventing transaction reordering manipulations in decentralized finance. *arXiv preprint arXiv:2203.11520*, 2022.

[Ili24] Danut Ilisei. Analyzing the role of bridges in cross-chain mev extraction. Master's thesis, TU München, 2024.

[JG24] Yan Ji and James Grimmelmann. Regulatory implications of mev mitigations. In *Proceedings of the 5th Workshop on the Coordination of Decentralized Finance*, 2024.

[@jm22] @jmcook.eth. Client diversity on ethereum's consensus layer. https://mirror.xyz/jmcook.eth/S7ONEka_0RgtKTZ3-dakPmAHQNPvuj15nh0YGKPFriA, Feb 2022.

[JSSW21] Aljosha Judmayer, Nicholas Stifter, Philipp Schindler, and Edgar Weippl. Estimating (miner) extractable value is hard, let's go shopping! *Cryptology ePrint Archive*, 2021.

[JvWR23] Johannes Rude Jensen, Victor von Wachter, and Omri Ross. Multi-block mev. *arXiv preprint arXiv:2303.04430*, 2023.

[KGF19] Rami Khalil, Arthur Gervais, and Guillaume Felley. Tex-a securely scalable trustless exchange. *Cryptology ePrint Archive*, 2019.

[KZGJ20] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III 40*, pages 451–480. Springer, 2020.

[Kö23] Martin Köppelmann. Reducing mev with transaction auction, Mar 2023.

[Lan21] Pi Lanningham. Sundaeswap fundamentals. *https://ouroboros.mobi/wp-content/uploads/2021/10/SundaeSwap-2021-06-01-Fundamentals.pdf*, 2021.

[LLPL24] Zekai Liu, Xiaoqi Li, Hongli Peng, and Wenkai Li. GasTrace: Detecting Sandwich Attack Malicious Accounts in Ethereum. Technical report, June 2024. arXiv:2405.19971 [cs] type: article.

[LZWD24] Dongze Li, Kejia Zhang, Lei Wang, and Gang Du. A Geth-based real-time detection system for sandwich attacks in Ethereum. *Discover Computing*, 27(1):11, May 2024.

[MKV23] Robert McLaughlin, Christopher Kruegel, and Giovanni Vigna. A large scale study of the ethereum arbitrage ecosystem. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3295–3312, 2023.

[MLTA09] David Moher, Alessandro Liberati, Jennifer Tetzlaff, and Douglas G Altman. Preferred reporting items for systematic reviews and meta-analyses: the prisma statement. *BMJ*, 339, 2009.

[MZVL24] Yifan Mao, Mengya Zhang, Shaileshh Bojja Venkatakrishnan, and Zhiqiang Lin. Flashback: Enhancing proposer-builder design with future-block auctions in proof-of-stake ethereum. *arXiv preprint arXiv:2405.09465*, 2024.

[Nak08] Satoshi Nakamoto. Bitcoin whitepaper. *bitcoin.org*, 2008.

[Noy21] Charlie Noyes. Mev and me. https://www.paradigm.xyz/2021/02/mev-and-me, Feb 2021.

[ÖKV+23] Burak Öz, Benjamin Kraner, Nicolò Vallarano, Bingle Stegmann Kruger, Florian Matthes, and Claudio Juan Tessone. Time moves faster when there is nothing you anticipate: The role of time in mev rewards. In *Proceedings of the 2023 Workshop on Decentralized Finance and Security*, pages 1–8, 2023.

[ÖRG+24] Burak Öz, Filip Rezabek, Jonas Gebele, Felix Hoops, and Florian Matthes. A study of mev extraction techniques on a first-come-first-served blockchain. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, pages 288–297, 2024.

[OSS+21] Alexandre Obadia, Alejo Salles, Lakshman Sankar, Tarun Chitra, Vaibhav Chellani, and Philip Daian. Unity is strength: A formalization of cross-domain maximal extractable value. *arXiv preprint arXiv:2112.01472*, 2021.

[PFW22] Julien Piet, Jaiden Fairoze, and Nicholas Weaver. Extracting godl [sic] from the salt mines: Ethereum miners extracting value. *arXiv preprint arXiv:2203.15930*, 2022.

[PJL+23] Seongwan Park, Woojin Jeong, Yunyoung Lee, Bumho Son, Huisu Jang, and Jaewook Lee. Unraveling the mev enigma: Abi-free detection model using graph neural networks. *arXiv preprint arXiv:2305.05952*, 2023.

[PMB$^+$21a] Matthew J Page, Joanne E McKenzie, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan, Roger Chou, Julie Glanville, Jeremy M Grimshaw, Asbjørn Hróbjartsson, Manoj M Lalu, Tianjing Li, Elizabeth W Loder, Evan Mayo-Wilson, Steve McDonald, Luke A McGuinness, Lesley A Stewart, James Thomas, Andrea C Tricco, Vivian A Welch, Penny Whiting, and David Moher. The prisma 2020 statement: an updated guideline for reporting systematic reviews. *BMJ*, 372, 2021.

[PMB$^+$21b] Matthew J Page, David Moher, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan, Roger Chou, Julie Glanville, Jeremy M Grimshaw, Asbjørn Hróbjartsson, Manoj M Lalu, Tianjing Li, Elizabeth W Loder, Evan Mayo-Wilson, Steve McDonald, Luke A McGuinness, Lesley A Stewart, James Thomas, Andrea C Tricco, Vivian A Welch, Penny Whiting, and Joanne E McKenzie. Prisma 2020 explanation and elaboration: updated guidance and exemplars for reporting systematic reviews. *BMJ*, 372, 2021.

[PRS$^+$06] Jennie Popay, Helen Roberts, Amanda Sowden, Mark Petticrew, Lisa Arai, Mark Rodgers, Nicky Britten, Katrina Roen, and Steven Duffy. *Guidance on the conduct of narrative synthesis in systematic reviews: A product from the ESRC Methods Programme.* Lancaster University, 01 2006.

[QZG$^+$21] Kaihua Qin, Liyi Zhou, Pablo Gamito, Philipp Jovanovic, and Arthur Gervais. An empirical study of defi liquidations: Incentives, risks, and instabilities. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 336–350, 2021.

[QZG22] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 198–214. IEEE, 2022.

[Rev22] Alvaro Revuelta. Multi-block mev in ethereum. https://www.alvarorevuelta.com/posts/ethereum-mev-multiblock, May 2022.

[RK20] Dan Robinson and Georgios Konstantopoulos. Ethereum is a dark forest, Aug 2020.

[RSW96] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.

[SHM21] Luke Van Seters, Guilherme Heise, and Robert Miller. Flashbots/mev-inspect-py: an mev inspector for ethereum, Jun 2021.

[SMC23] Johan Hagelskjar Sjursen, Weizhi Meng, and Wei-Yang Chiu. Towards quantifying cross-domain maximal extractable value for blockchain decentralisation. In *International Conference on Information and Communications Security*, pages 627–644. Springer, 2023.

[TC+21] Christof Ferreira Torres, Ramiro Camino, et al. Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1343–1359, 2021.

[TMW+24] Christof Ferreira Torres, Albin Mamuti, Ben Weintraub, Cristina Nita-Rotaru, and Shweta Shinde. Rolling in the shadows: Analyzing the extraction of mev across layer-2 rollups. *arXiv preprint arXiv:2405.00138*, 2024.

[VNSA+21] Vbuterin, Nazariyv, Shymaa-Arafat, Jannikluhn, Yoavw, JustinDrake, and Lekssays. Proposer/block builder separation-friendly fee market designs, Jun 2021.

[vro23] vrotend. Mev-aware application design: Great visual example! https://x.com/vrotend/status/1637106929083338752, March 2023. [Tweet].

[W+14] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[WH24] Matthew Willetts and Christian Harrington. Multiblock mev opportunities & protections in dynamic amms. *arXiv preprint arXiv:2404.15489*, 2024.

[WTNRS22] Ben Weintraub, Christof Ferreira Torres, Cristina Nita-Rotaru, and Radu State. A flash (bot) in the pan: measuring maximal extractable value in private pools. In *Proceedings of the 22nd ACM Internet Measurement Conference*, pages 458–471, 2022.

[Wun23] Sebastian Wunderlich. Exploring Maximal Extractable Value in the Ethereum Ecosystem. *https://blockchain.hs-mittweida.de/wordpress/wp-content/uploads/2023/08/WunderlichSebastianMasterFinal-1.pdf*, 2023.

[XPCF23] Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *ACM Computing Surveys*, 55(11):1–50, 2023.

[YHL+] Zihao Yao, Fanding Huang, Yannan Li, Wei Duan, Peng Qian, Nan Yang, and Willy Susilo. Mecon: A Gnn-Based Graph Classification Framework for Mev Activity Detection. *Available at SSRN 4861523.*

[YLK+24] Tao Yan, Shengnan Li, Benjamin Kraner, Luyao Zhang, and Claudio J Tessone. Analyzing reward dynamics and decentralization in ethereum 2.0:

An advanced data engineering workflow and comprehensive datasets for proof-of-stake incentives. *arXiv preprint arXiv:2402.11170*, 2024.

[YZH+22] Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. Sok: Mev countermeasures: Theory and practice. *arXiv preprint arXiv:2212.05111*, 2022.

[ZNW21] Patrick Züst, Tejaswi Nadahalli, and Ye Wang Roger Wattenhofer. Analyzing and preventing sandwich attacks in ethereum. *ETH Zürich*, pages 1–29, 2021.

[ZQC+21] Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, and Arthur Gervais. On the just-in-time discovery of profit-generating transactions in defi protocols. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 919–936. IEEE, 2021.

[ZQG21] Liyi Zhou, Kaihua Qin, and Arthur Gervais. A2mm: Mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges. *arXiv preprint arXiv:2106.07371*, 2021.

Appendix

Table 1: The PRISMA 2020 Checklist [PMB+21a] used in the systematic Review in chapter 4

Section and Topic	Item #	Checklist item	Location where item is reported
TITLE			
Title	1	Identify the report as a systematic review.	chapter 4
ABSTRACT			
Abstract	2	See the PRISMA 2020 for Abstracts checklist.	chapter 4
INTRODUCTION			
Rationale	3	Describe the rationale for the review in the context of existing knowledge.	chapter 4 - Introduction
Objectives	4	Provide an explicit statement of the objective(s) or question(s) the review addresses.	chapter 4 - Introduction
METHODS			
Eligibility criteria	5	Specify the inclusion and exclusion criteria for the review and how studies were grouped for the syntheses.	subsection 4.1.1
Information sources	6	Specify all databases, registers, websites, organisations, reference lists and other sources searched or consulted to identify studies. Specify the date when each source was last searched or consulted.	subsection 4.1.2
Search strategy	7	Present the full search strategies for all databases, registers and websites, including any filters and limits used.	subsection 4.1.3

Section and Topic	Item #	Checklist item	Location where item is reported
Selection process	8	Specify the methods used to decide whether a study met the inclusion criteria of the review, including how many reviewers screened each record and each report retrieved, whether they worked independently, and if applicable, details of automation tools used in the process.	subsection 4.1.4
Data collection process	9	Specify the methods used to collect data from reports, including how many reviewers collected data from each report, whether they worked independently, any processes for obtaining or confirming data from study investigators, and if applicable, details of automation tools used in the process.	subsection 4.1.5
Data items	10a	List and define all outcomes for which data were sought. Specify whether all results that were compatible with each outcome domain in each study were sought (e.g. for all measures, time points, analyses), and if not, the methods used to decide which results to collect.	subsection 4.1.5
	10b	List and define all other variables for which data were sought (e.g. participant and intervention characteristics, funding sources). Describe any assumptions made about any missing or unclear information.	subsection 4.1.5
Study risk of bias assessment	11	Specify the methods used to assess risk of bias in the included studies, including details of the tool(s) used, how many reviewers assessed each study and whether they worked independently, and if applicable, details of automation tools used in the process.	subsection 4.1.6

Section and Topic	Item #	Checklist item	Location where item is reported
Effect measures	12	Specify for each outcome the effect measure(s) (e.g. risk ratio, mean difference) used in the synthesis or presentation of results.	subsection 4.1.6
Synthesis methods	13a	Describe the processes used to decide which studies were eligible for each synthesis (e.g. tabulating the study intervention characteristics and comparing against the planned groups for each synthesis (item #5)).	subsection 4.1.6
	13b	Describe any methods required to prepare the data for presentation or synthesis, such as handling of missing summary statistics, or data conversions.	subsection 4.1.6
	13c	Describe any methods used to tabulate or visually display results of individual studies and syntheses.	subsection 4.1.6
	13d	Describe any methods used to synthesize results and provide a rationale for the choice(s). If meta-analysis was performed, describe the model(s), method(s) to identify the presence and extent of statistical heterogeneity, and software package(s) used.	subsection 4.1.6
	13e	Describe any methods used to explore possible causes of heterogeneity among study results (e.g. subgroup analysis, meta-regression).	subsection 4.1.6
	13f	Describe any sensitivity analyses conducted to assess robustness of the synthesized results.	subsection 4.1.6
Reporting bias assessment	14	Describe any methods used to assess risk of bias due to missing results in a synthesis (arising from reporting biases).	n/a
Certainty assessment	15	Describe any methods used to assess certainty (or confidence) in the body of evidence for an outcome.	subsection 4.1.6
RESULTS			

Section and Topic	Item #	Checklist item	Location where item is reported
Study selection	16a	Describe the results of the search and selection process, from the number of records identified in the search to the number of studies included in the review, ideally using a flow diagram.	subsection 4.1.7
	16b	Cite studies that might appear to meet the inclusion criteria, but which were excluded, and explain why they were excluded.	subsection 4.1.8
Study characteristics	17	Cite each included study and present its characteristics.	section 4.2
Risk of bias in studies	18	Present assessments of risk of bias for each included study.	n/a
Results of individual studies	19	For all outcomes, present, for each study: (a) summary statistics for each group (where appropriate) and (b) an effect estimate and its precision (e.g. confidence/credible interval), ideally using structured tables or plots.	subsection 4.2.1
Results of syntheses	20a	For each synthesis, briefly summarise the characteristics and risk of bias among contributing studies.	n/a
	20b	Present results of all statistical syntheses conducted. If meta-analysis was done, present for each the summary estimate and its precision (e.g. confidence/credible interval) and measures of statistical heterogeneity. If comparing groups, describe the direction of the effect.	n/a
	20c	Present results of all investigations of possible causes of heterogeneity among study results.	n/a
	20d	Present results of all sensitivity analyses conducted to assess the robustness of the synthesized results.	n/a
Reporting biases	21	Present assessments of risk of bias due to missing results (arising from reporting biases) for each synthesis assessed.	n/a

Section and Topic	Item #	Checklist item	Location where item is reported
Certainty of evidence	22	Present assessments of certainty (or confidence) in the body of evidence for each outcome assessed.	n/a
DISCUSSION			
Discussion	23a	Provide a general interpretation of the results in the context of other evidence.	subsection 4.2.1
	23b	Discuss any limitations of the evidence included in the review.	subsection 4.2.1
	23c	Discuss any limitations of the review processes used.	subsection 4.2.1
	23d	Discuss implications of the results for practice, policy, and future research.	subsection 4.2.1
OTHER INFORMATION			
Registration and protocol	24a	Provide registration information for the review, including register name and registration number, or state that the review was not registered.	n/a
	24b	Indicate where the review protocol can be accessed, or state that a protocol was not prepared.	Table 1
	24c	Describe and explain any amendments to information provided at registration or in the protocol.	section 4.1
Support	25	Describe sources of financial or non-financial support for the review, and the role of the funders or sponsors in the review.	n/a
Competing interests	26	Declare any competing interests of review authors.	n/a
Availability of data, code and other materials	27	Report which of the following are publicly available and where they can be found: template data collection forms; data extracted from included studies; data used for all analyses; analytic code; any other materials used in the review.	n/a

Queries

```sql
WITH RECURSIVE sequences AS (
    SELECT
        inital_b.id,
        inital_b.number,
        val.label,
        1 AS sequence_length
    FROM
        block inital_b
    JOIN
        validator val ON inital_b.validator_index = val.index
    JOIN block previous_b ON inital_b.number = previous_b.number + 1
    JOIN validator previous_val ON previous_b.validator_index = previous_val.
        index
    WHERE inital_b.not_in_mev_boost = True AND
    val.label <> previous_val.label -- to make sure we start at the start of
        the sequence, not the nth element in a sequence


    UNION ALL

    SELECT
        t.id,
        t.number,
        v.label,
        s.sequence_length + 1 AS sequence_length
    FROM
        block t
    JOIN
        validator v ON t.validator_index = v.index
    JOIN
        sequences s ON t.number = s.number + 1
    WHERE
      t.not_in_mev_boost = True
      AND v.label = s.label
)
SELECT sequence_length, COUNT(*) FROM (
SELECT
    id,
    number,
    label,
    sequence_length
FROM
    sequences
WHERE sequence_length > 1
) grouped GROUP BY grouped.sequence_length
ORDER BY sequence_length DESC
```

Listing 1: Finding sequences in blocks not proposed through MEV-Boost that have the same pool

```sql
WITH RECURSIVE sequences AS (
    SELECT
        id,
        block_number,
        builder,
        mevboost_value_num::numeric,
        1 AS sequence_length
    FROM
        mev_boost_pics_data_validated

    UNION ALL

    SELECT
        t.id,
        t.block_number,
        t.builder,
        (t.mevboost_value_num + s.mevboost_value_num) AS mevboost_value_num,
        s.sequence_length + 1 AS sequence_length
    FROM
        mev_boost_pics_data_validated t
    JOIN
        sequences s ON t.block_number = s.block_number + 1
                 AND t.builder = s.builder
)
SELECT PERCENTILE_CONT(0.5)
WITHIN GROUP(ORDER BY s1.mevboost_value_num)
FROM    sequences s1
WHERE s1.sequence_length = 11
-- ORDER BY s1.sequence_length DESC, s1.mevboost_value_num DESC
LIMIT 100
```

Listing 2: Calculating the median values of blocks in consecutive sequences built with MEV-Boost

```sql
SELECT
CONCAT(
        SUBSTRING(token_in_address, 1, 5),
        '...',
        SUBSTRING(token_in_address, LENGTH(token_in_address) - 3, 4)
    ) AS token_in_address,
CONCAT(
        SUBSTRING(token_out_address, 1, 5),
        '...',
        SUBSTRING(token_out_address, LENGTH(token_out_address) - 3, 4)
    ) AS token_out_address,
 token_in_amount, token_out_amount, trace_address
FROM "swaps_liquidations_merged"
WHERE transaction_hash = '0
    x350ca80a3db7d792d37a200933d88a3dfd472cfdd2cd11f127d73f26767d640c'
ORDER BY trace_address ASC
LIMIT 50
```

Listing 3: SQL Query used to select the tokenflow of the transaction 0x350...d640c