I See You! Robust Measurement of Adversarial Behavior

Lars L. Ankile
larsankile@g.harvard.edu
Harvard University

Matheus V. X. Ferreira
matheus@seas.harvard.edu
Harvard University

David C. Parkes
parkes@eecs.harvard.edu
Harvard University

Abstract

We introduce the study of non-manipulable measures of manipulative behavior in multi-agent systems. We do this through a case study of decentralized finance (DeFi) and blockchain systems, which are salient as real-world, rapidly emerging multi-agent systems with financial incentives for malicious behavior, for the participation in algorithmic and AI systems, and for the need for new methods with which to measure levels of manipulative behavior. We introduce a new surveillance metric for measuring malicious behavior and demonstrate its effectiveness in a natural experiment to the Uniswap DeFi ecosystem. Find code and data here.

1 Introduction

As AI and multi-agent systems rapidly evolve, a pressing challenge is developing reliable metrics for measuring manipulative behavior within these systems—*can we develop non-manipulable measures of the level of manipulative behavior in a multi-agent system?* This is crucial for fostering robust and trustworthy decentralized environments.

This paper uses decentralized finance (DeFi) and blockchain as case studies. These systems offer a pertinent context due to their rapid emergence, clear incentives for malicious behavior, involvement of algorithmic and AI systems, and the need for new methods to quantify manipulative actions (and, in turn, the need for interventions to promote trustworthy behavior).

Decentralized exchanges such as Uniswap process over $1 billion in daily trading volume and are primary applications for permissionless blockchains. Blockchains are open systems where anyone can join and participate, and decentralized exchanges have inherent security vulnerabilities and face attacks that require novel approaches to detection. As researchers, the transparency of decentralized exchanges is helpful, as much of the trading data is public, enabling research into data-driven techniques to detect manipulation.

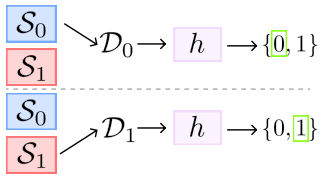


Figure 1: Bundles of transactions $\mathcal{D}_b$ originate from one of two sources, $\mathcal{S}_0$ (altruistic sequencer) or $\mathcal{S}_1$ (malicious sequencer). Classifier h predicts source.

We study *detection algorithms* for DeFi from the lens of an external observer who must detect price manipulation given only the sequences of transactions or, equivalently, the price trajectories of a particular asset. The focus is on the role of *sequencers*, who receive transactions from a communication network and decide what order the exchange executes them and may also inject and censor transactions. Concretely, we formulate a *classification problem*, Figure 1, of detecting whether a communication network is

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

connected to a malicious sequencer. A sequencer belongs to the class of *altruistic sequencers* $\mathcal{S}_0$ if they do not manipulate transaction orders and belongs to the type of *malicious sequencers*, $\mathcal{S}_1$ if they exploit traders. Given a database of execution orderings $\mathcal{D}_b$ generated by a particular sequencer, the task of h is to classify the sequencer as altruistic (i.e., $b = 0$) or malicious (i.e., $b = 1$).

Our approach defines a ***Surveillance metric*** S that maps an execution ordering T to a score $S(T)$. We ask if there is a surveillance metric that correlates with a high likelihood that a sequencer is malicious. We summarize our findings as follows:

1. We introduce a new surveillance metric, Definition 1, and demonstrate by a natural experiment that sequencers known to exploit traders have a higher surveillance metric than altruistic sequencers.
2. We argue that the metric is robust to obfuscation from an adversary using multiple identities or more complex strategies. We provide a basis for this working hypothesis that the metric correlates with a higher likelihood of sequencer manipulation.

Paper Organization Section 2 gives a brief introduction to DeFi. Section 3 defines the surveillance metric S. Section 4 gives a motivating example of a sandwich attack. Section 5 gives the experimental results. We conclude in Section 6.

2 Preliminaries: The DeFi Multi-Agent System

This section briefly describes the communication game between traders and a decentralized exchange such as Uniswap. An exchange has a *state* X_t, at time $t \geq 1$, and executes transactions sequentially. In the case of Uniswap, X_t represents the capital liquidity providers have "locked" into a contract. The *communication game* proceeds as follows:

1. Each *trader* i sends a *transaction* T_i to a *communication network*. For simplicity, assume a transaction is either a buy order, $\textsc{Buy}(q, p)$, that purchases q units of the asset at a price at most p US Dollars or a sell order, $\textsc{Sell}(q, p)$, that sells q units of the asset at a price at least p US Dollars. We write $\textsc{Buy}(q) = \textsc{Buy}(q, \infty)$ and $\textsc{Sell}(q) = \textsc{Sell}(q, 0)$.
2. A *sequencer* connects to the communication network and observes transactions $T = \{T_1, \ldots, T_n\}$. The sequencer outputs some transaction ordering $(T_{\sigma_1}, \ldots, T_{\sigma_n})$ where $(\sigma_1, \ldots, \sigma_n)$ is a permutation of $(1, \ldots n)$. The transaction ordering a sequencer outputs differs for each exchange. For example, a first-come-first-served sequencer would order transactions by time—if the sequencer observes T_i before T_j, then $\sigma_i < \sigma_j$ in the message ordering.
3. The *exchange* receives $(T_{\sigma_1}, \ldots, T_{\sigma_n})$ from the sequencer and consumes messages in this order; i.e., it first consumes T_{σ_1} at state X_1 and transitions to state X_2, then it consumes T_{σ_2} at state X_2 and changes to state X_3, and so on.

We focus on detecting malicious sequencers that observe T and can choose any transaction ordering $(T'_1, \ldots, T'_k)$ that contain transactions from T and may also include additional trades of their own. An attack (or malicious behavior) happens under the following scenarios:

- **Message injection.** If $T'_i \notin S$, then T'_i is a transaction the sequencer injects in the transaction order. We are mainly concerned with the case where the sequencer uses their knowledge about T to generate a transaction T'_i that depends on S, which suggests the sequencer is using their privileged position to profit from unsuspecting traders.
- **Message deletion.** If $T_i \notin T'_j$ for any j, then the sequencer censored T_i from trader i.
- **Message reordering.** Given a set of transactions $\{T'_1, \ldots, T'_k\}$, the sequencer picks any ordering $(T'_1, \ldots, T'_k)$ they wish.

A *sandwich attack* is an especially salient example of an attack on a decentralized exchanges (Example 1). A sandwich attach results in risk-free profits for the sequencer and contributes to a phenomenon known as *Maximal Extractable Value* (MEV) [2]—which refers to the profits that autonomous agents can "extract" from blockchain systems.

Example 1 (Sandwich attack)**.** *Suppose a single trader sends a buy order* $\textsc{Buy}(q)$*. In the attack, the sequencer picks the ordering* $(\textsc{Buy}(w), \textsc{Buy}(q), \textsc{Sell}(w))$ *where* $\{\textsc{Buy}(w), \textsc{Sell}(w)\}$ *are the*

attacker's transactions. $\text{BUY}(w)$ *drives the asset's price higher. This induces* $\text{BUY}(q)$ *to the trade at a higher price, increasing the asset's price a second time. Finally, after* $\text{SELL}(w)$ *executes, the attacker sells* w *units of the token at a higher price than they bought, yielding risk-free profits.*

Sandwich attacks can be detected via *rule-based mechanisms* [13, 4, 8]. For example, if an execution ordering contains the pattern $(\ldots, \text{BUY}(w), \text{BUY}(q), \text{SELL}(w), \ldots)$ where $\text{BUY}(w)$ and $\text{SELL}(w)$ are transactions from the same trader, then we might flag the communication network as exploiting trade $\text{BUY}(q)$. Traders have options regarding the communication networks through which they transmit their transactions and could use this simple detection mechanism to avoid risky networks.

However, these kinds of rule-based detection mechanisms are easy to obfuscate. In our example, if $\text{BUY}(w)$ and $\text{SELL}(w)$ are transactions from different traders controlled by our attacker, then the attack would pass undetected (See Appendix B for additional rule-based mechanisms and ways attackers can circumvent them). Rule-based surveillance systems are not unique to DeFi. Financial Industry Regulatory Authority (FINRA) surveillance program traditionally relies on rule-based programming and, due to their limitations, there is interest in adopting data-driven approaches [7].

3 The Surveillance Metric

We now introduce our surveillance metric, which measures how much prices deviate from the initial price $p(\emptyset)$, defined as the asset price before any transaction executes. Formally, let $p(T)$ be the asset price after transactions in T execute, and define $T_{\leq i} := (T_1, \ldots, T_i)$ as the execution ordering of the first i transactions in T.

Definition 1 (p-Surveillance Metric)**.** *The p-surveillance metric for $p \geq 1$ is* $S_p(T) = (\sum_{i=1}^{n} |p(T_{\leq i}) - p(\emptyset)|^p)^{\frac{1}{p}}$.

There are instances where the surveillance metric will always be large for any permutation of T. For example, consider a case with n identical buy orders $\text{BUY}(q)$. Any permutation of these transactions will have the same surveillance metric, and this grows proportionally with n.

Thus, we also normalize the metric as follows. Formally, let the *optimal execution ordering* T^* be the permutation of T that minimizes the surveillance metric, that is $T^* \in \arg\min_{T'} S_p(T')$.

Given this, the *normalized surveillance metric* is defined as $\bar{S}_p(T) := \frac{S_p(T)}{S_p(T^*)} - 1$. We devise a polynomial time heuristic in Appendix D to approximate T^*.

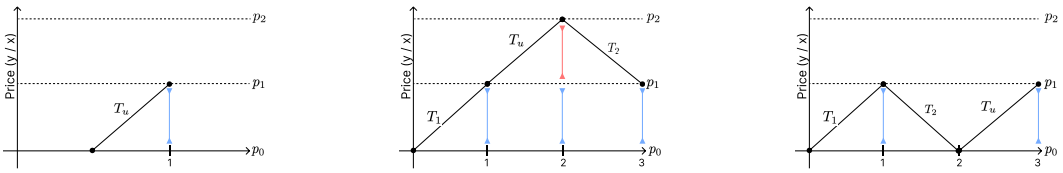
4 Motivating Example: Surveillance Metric on the Sandwich Attack

The surveillance metric quantifies how much prices deviate from the initial price before any transaction is executed. This section uses a sandwich attack to explain why this metric correlates with a sequencer manipulating the execution order.

Recall that a trader u will communicate a transaction $T_u = \text{BUY}(1)$ in a sandwich attack. If the execution ordering is (T_u), the asset price increases from p_0 to p_1 Figure 2a. Suppose instead the adversary picks the execution ordering $(\text{BUY}(2), T_u, \text{SELL}(2))$. After $(\text{BUY}(2), T_u, \text{SELL}(2))$ execute, Figure 2b, the asset price increases first from p_0 to $p_1 > p_0$, then from $p_2 > p_1$, and finally decreases from p_2 to p_1. Although the final asset price is the same on both execution orderings (T_u) and $(\text{BUY}(2), T_u, \text{SELL}(2))$, u gets a worse price, and the sequencer receives risk-free profits.

When we reorder the execution ordering $(\text{BUY}(2), T_u, \text{SELL}(2))$ to minimize the surveillance metric, we obtain $(\text{BUY}(2), \text{SELL}(2), T_u)$, Figure 2c. The metric minimizing ordering causes no harm to u since it receives the same execution price as in the execution ordering (T_u). This example motivates analyzing deviations from the baseline price p_0.

Observe our surveillance metric is invariant to an adversary that obfuscates an attack by (1) having multiple trader IDs and (2) splitting a transaction into multiple smaller transactions. For example, $\text{BUY}(2)$ and $\text{SELL}(2)$ could be transactions from different trader IDs the adversary controls. Moreover, the adversary can split $\text{BUY}(2)$ into k transactions $\text{BUY}(x_1), \ldots, \text{BUY}(x_k)$ such that $\sum_{i=1}^{k} = 2$ (potentially with each $\text{BUY}(x_i)$ coming from a different trader ID). Our surveillance metric is invariant to either transformation. See Appendix C for another example and failure mode.



(a) Standalone execution of T_u

(b) Sandwich attack

(c) Alternate permutation

Figure 2: (a) Fair transaction ordering. (b) Manipulated transaction ordering. (c) The transaction ordering that minimizes the surveillance metric.

5 Experimental Results: A Natural Experiment on Ethereum

In this section, we report the results of a *natural experiment* in the Uniswap DeFi ecosystem on the Ethereum blockchain. In our empirical study, execution orderings originate from one of two sources. This natural experiment takes advantage of *MEV-boost*. This technology allows blockchain operators to implement an auction where autonomous agents can compete to be the sequencer. Although around 85% (Figure 10, [14]) of transactions are chosen via MEV-boost, blockchain operators are not required to auction their transaction sequencing on this auction.

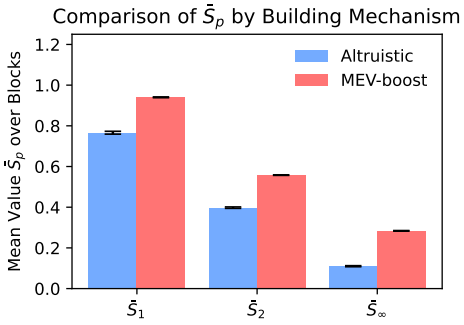


Figure 3: Blocks built through the MEV-boost auction mechanism show a significant increase in our measure $\bar{S}_p$ compared to blocks built by altruistic sequencers. The mean is calculated by finding $\bar{S}_p$ for each block and taking the mean value over blocks within groups.

Consider a malicious transaction sequencer. They are likely to be more competitive on the MEV-boost auction because they can redirect part of their profits from manipulation to bids on the auction. Conversely, altruistic sequencers are likely less competitive on MEV-boost. Indeed, sequencers participating in MEV-boost are believed to extract value from exploiting the execution price of vulnerable traders [9]. Thus, execution orderings originating from an MEV-boost sequencer provide our first source of sequenced transactions. All remaining execution orderings constitute our second source. Our working hypothesis (that we confirm) is that transaction orderings coming from the first source will have higher surveillance metrics than in the second source.

We experiment with Uniswap trading data to validate that sequencers participating in MEV boost have higher surveillance metrics. We collect all Uniswap V3 transactions on the Ethereum blockchain since the launch of the MEV-boost auction, Sept. 15th, 2022, to Aug. 7th, 2023, 1,866,537 blocks and 7,994,736 trades in total.[1] For our experiments, we compute $\bar{S}_p(T)$ for $p = 1, 2$, and ∞ for each execution ordering T that contains three or more transactions. We average the normalized surveillance metric for execution orderings from the same source and plot the results in Figure 3. The results show the average surveillance metric is substantially higher for sequencers who participate in MEV-boost than those who do not (and this difference is highly statistically significant, Appendix G). This holds for our choices of p. This validates our hypothesis that the surveillance metric is a valuable measure of manipulative behavior in this complex blockchain ecosystem. We present further analysis in Appendix F.

6 Conclusion

With the rise of multi-agent systems involving economic incentives, it is invaluable to develop tools to detect manipulation. We take Uniswap DeFi as a case study due to clear evidence of manipulative behavior. We propose a surveillance metric to detect price manipulations in Uniswap that is robust

[1] See Appendix E for details. We will make the full dataset and all code publicly available in the final version.

to an adversary attempting to obfuscate an attack. We validate our metric on a natural experiment and show it can distinguish altruistic and malicious transaction sequencers with high confidence. Our surveillance mechanism provides a quantitative measure of the credibility of DeFi by allowing external observers to audit [5]. Our inquiry leaves many experimental and theoretical questions. Theoretically, one could ask for sufficient conditions for a surveillance metric to be non-decreasing with an adversary's utility. That is particularly relevant because, although an adversary's utility is non-observable, any external observer can compute our surveillance metric.

Acknowledgments

We thank Austin Adams and Xin Wan from Uniswap Labs for their invaluable help with data collection and deciphering.

References

[1] M. Allwood. The satterthwaite formula for degrees of freedom in the two-sample t-test. *The College Board*, 2008.

[2] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. *Proceedings - IEEE Symposium on Security and Privacy*, 2020-May:1106–1120, May 2020. ISSN 10816011. doi: 10.1109/SP40000.2020.00040. Publisher: Institute of Electrical and Electronics Engineers Inc. ISBN: 9781728134970.

[3] B. Derrick, D. Toher, and P. White. Why welch's test is type i error robust. *The quantitative methods for Psychology*, 12(1):30–38, 2016.

[4] EigenPhi. Recognizing Cross-Transaction Sandwich MEV, May 2023. URL https://eigenphi-1.gitbook.io/classroom/eigenphis-methodologies/how-eigenphi-identifies-mev/recognizing-cross-transaction-sandwich-mev.

[5] M. V. X. Ferreira and D. C. Parkes. Credible Decentralized Exchange Design via Verifiable Sequencing Rules. *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, 2023. arXiv: 2209.15569v2.

[6] M. V. X. Ferreira and S. M. Weinberg. Proof-of-Stake Mining Games with Perfect Randomness. In *Proceedings of the 22nd ACM Conference on Economics and Computation*, pages 433–453, July 2021. doi: 10.1145/3465456.3467636. URL http://arxiv.org/abs/2107.04069. arXiv:2107.04069 [cs, econ].

[7] FINRA, Jun 2022. URL https://www.finra.org/sites/default/files/2020-06/ai-report-061020.pdf.

[8] Flashbots. Maximal extractable value inspector for Ethereum, to illuminate the dark forest, 2021. URL https://github.com/flashbots/mev-inspect-py.

[9] Flashbots. Flashbots mev-boost, 2023. URL https://github.com/flashbots/mev-boost#how-does-mev-boost-work.

[10] K. Qin, L. Zhou, and A. Gervais. Quantifying Blockchain Extractable Value: How dark is the forest? *Proceedings - IEEE Symposium on Security and Privacy*, 2022-May:198–214, 2022. ISSN 10816011. doi: 10.1109/SP46214.2022.9833734. arXiv: 2101.05511 Publisher: Institute of Electrical and Electronics Engineers Inc. ISBN: 9781665413169.

[11] G. D. Ruxton. The unequal variance t-test is an underused alternative to student's t-test and the mann–whitney u test. *Behavioral Ecology*, 17(4):688–690, 2006.

[12] D. S. Starnes, D. Yates, and D. S. Moore. *The practice of statistics*. Macmillan, 2010.

[13] C. F. Torres, R. Camino, and R. State. Frontrunner Jones and the Raiders of the Dark Forest: An Empirical Study of Frontrunning on the Ethereum Blockchain. *30th USENIX Security Symposium (USENIX Security 21)*, Feb. 2021. URL http://arxiv.org/abs/2102.03347. arXiv: 2102.03347.

[14] A. Wahrstätter, L. Zhou, K. Qin, D. Svetinovic, and A. Gervais. Time to Bribe: Measuring Block Construction Market. *Cryptology ePrint Archive, Paper 2023/760*, May 2023. arXiv: 2305.16468.

[15] Y. Wang wangye, e. ETH Zurich Zurich, S. Patrick Zuest, E. Zurich Zurich, S. Yaxing Yao, Z. Lu, R. Wattenhofer wattenhofer, Y. Wang, P. Zuest, Y. Yao, and R. Wattenhofer. Impact and User Perception of Sandwich Attacks in the DeFi Ecosys-tem. *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022. doi: 10.1145/3491102.3517585. URL https://doi.org/10.1145/3491102.3517585. ISBN: 9781450391573.

[16] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais. High-frequency trading on decentralized on-chain exchanges. *Proceedings - IEEE Symposium on Security and Privacy*, 2021-May:428–445, May 2021. ISSN 10816011. doi: 10.1109/SP40001.2021.00027. arXiv: 2009.14021 Publisher: Institute of Electrical and Electronics Engineers Inc. ISBN: 9781728189345.

[17] P. Züst, T. Nadahalli, and Y. Wang Roger Wattenhofer. Analyzing and Preventing Sandwich Attacks in Ethereum. *ETH Zürich*, 2021. URL www.DeFi-Sandwi.ch.

A Related Work

Besides the AMM DEXs considered in this work, many other decentralized exchanges, e.g., limit order book-based, auction-based, off-chain matching, and payment channels, exist and are addressed in the literature[16]. AMMs are still the most prevalent mechanism for trading on the blockchain.

Daian et al. [2] coined the term Miner Extractable Value (MEV), focusing on Priority Gas Auctions (PGA) for exploiting unconditional profit through atomic transactions. With the transition to Proof-of-Stake (PoS) Ethereum and the introduction of Proposer-Builder Separation (PBS), PGAs have been replaced by Flashbots' MEV-boost auctions. Sandwich attacks, central to this study, emerged as revenue opportunities within MEV-boost auctions, as they cannot be executed atomically within a single transaction.

Zhou et al. [16] offered a formal definition and early quantification of sandwich attacks, along with estimating the attackers' profit potential. Before PBS, these attacks required complex PGAs, where attackers strategically outbid and underbid victims and other adversaries. Zhou et al. [16] identified minimum victim transaction sizes of approximately 22 ETH and 15 ETH for Uniswap's ETH/SAI and ETH/DAI pools, respectively. They conducted an empirical analysis by establishing an adversarial Ethereum node that monitored the mempool for victim transactions, focusing on those generated by themselves, and executed attacks accordingly.

Wang wangye et al. [15] analyze the presence of sandwich attacks from the victims' perspective, i.e., regular users, thus focusing on the losses they incur instead of the gains the attackers accrue. They show empirically that from May 2020 to April 2021, the absolute number of sandwich attacks increased meaningfully, and the proportion of potential sandwich attacks exploited increased.

Torres et al. [13] empirically analyzed various attack vectors (displacement, insertion, and suppression) from 2018 to 2020 before Uniswap V3 and PoS Ethereum. Their work highlighted the prevalence of insertion attacks in Uniswap V2 and the challenges attackers faced in controlling frontrunning and backrunning transactions through gas prices in PGAs.

Qin et al. [10] performed a 32-month-long study of Blockchain Extracted Value (BEV), i.e., what we term MEV. The study considered the attack vectors of sandwiches, liquidations, arbitrage, and transaction replay. They estimate BEV-extraction in the period to amount to $541M, with sandwich attacks contributing $174M of them. As most other retrospective measurements of sandwich attack revenue, they rely on rules or heuristics to detect explicit attacks.

Wahrstätter et al. [14] explored the changes in the block builder market after Ethereum's transition from Proof-of-Work (PoW) to PoS in September 2022, enabling PBS. They argued that PBS, despite its decentralization goals, might have introduced more implicit trust assumptions. Under PBS, the Ethereum blockchain involves three main actors: Searchers, Builders, and Relays, with MEV-boost having a dominant share in block proposing. The study found that block proposers were compensated approximately 160,000 ETH during the study period.

B Summary of other Sandwich Attack Detection Methods

In this section, we use the term *frontrunning* to mean any transaction inserted by an adversary such that it executes on the exchange before the victim's transaction executes. The opposite case, where an adversary inserts a trade after the victim, is called *backrunning*. The combination of a frontrun and backrun attack is known as a *sandwich attack*. All of the below methods are designed to detect sandwich attacks.

Furthermore, in the context of the Ethereum-based exchange Uniswap,[2] the most popular Decentralized Exchange (DEX) currently, each market for a specific asset is called a *pool*. This comes from the fact that markets function by letting users trade against a reserve of assets locked in the exchange instead of having an explicit counterpart. This reserve is called a liquidity pool.

As on standard stock exchanges, any trade incurs trading fees. In the context of the Ethereum blockchain, these fees are called *gas fees*. Gas fees incur whenever a computation happens on the blockchain. So, as an example, a complex arbitrage strategy involving many trades will incur more gas fees than a simple trade.

[2] https://docs.uniswap.org/

Lastly, when trading on the blockchain, there is no notion of Know Your Customer (KYC) as in traditional finance. Instead, all participants are assigned a random and unique ID called an *address*. One person can create as many addresses as they want very cheaply.

B.1 Summary of Limitations of Current Detection Methods

We examine various methods for detecting sandwich attacks on the Ethereum blockchain, focusing on Flashbots' `MEV-inspect-py` due to its prominence in the ecosystem. The code and heuristics for these methods are detailed in Appendix B. Flashbots' approach has three key limitations (Appendix B.2):

1. **Address Matching:** Attackers can evade detection by using different addresses (accounts) for frontrunning and backrunning and rebalance between them in a later block.
2. **Indirect Calls:** The method excludes transactions to Uniswap router contracts, as trading through the router can incur higher fees. However, the gas cost is often not prohibitive.
3. **Volume Splitting:** By breaking the frontrunning or backrunning swap into smaller transactions, attackers can make a profitable sandwich attack go undetected or appear unprofitable if it is.

Other models, such as those by EigenPhi [4] and Züst et al. [17], share similarities with Flashbots and are discussed in Appendix B.3. The model by Torres et al. [13] employs heuristics like address and amount matching but allows for a 1% variance in amounts (Appendix B.4). Qin et al. [10] is more tolerant of volume discrepancies, allowing up to a 10% difference between frontrunning and backrunning transactions. A common assumption in these models is the presence of a single victim transaction, offering another avenue for attack obfuscation. Another interesting hidden assumption in all methods studies is the implicit choice of utility function for the attacker (the subject of future work).

Incentives for Obfuscation: Since blockchains are unregulated, adversarial behavior is not prosecuted. Still, some actors, such as wallet providers and trusted block builders, may have strong incentives to obscure their malicious activities to maintain user trust.

B.2 Flashbots sandwich detection code

This part contains a distillation of the detection code in the `MEV-inspect-py` package Flashbots uses to calculate their estimates of MEV extracted. Please see the documentation[3] and GitHub code[4] for more.

In short, their heuristics-based approach relies on these rules:

1. If the suspected frontrunning swap comes from a router contract,[5] abort detection. This is likely done to make attribution easier and is probably deemed reasonable because the attacker will incur slightly higher fees by employing this obfuscation tactic.
2. Frontrunning, victim, and backrunning transactions must be sent to the same liquidity pool.
3. For a transaction to be counted as a victim transaction, it must happen after the frontrunning transaction and be in the same direction, i.e., be a buy transaction if the frontrunning transaction buys.
4. For a transaction to be counted as a backrunning transaction, there has to be at least one victim, and it needs to be in the opposite direction of the frontrunning transaction and needs to have the same address as the frontrunning swap. At the first swap that satisfies these conditions, the procedure terminates.
5. The estimated profit of the attack is calculated as the difference between the assets the attacker got out in the backrunning transaction and the assets they put in in the frontrunning transaction.

[3] https://docs.flashbots.net/flashbots-data/mev-inspect-py/overview

[4] https://github.com/flashbots/mev-inspect-py/tree/main

[5] Essentially code that takes an order to any trading pool and sends it (reroutes it) to the correct market. See more at https://docs.uniswap.org/contracts/universal-router/overview.

Most of these rules are reasonable and not possible to fool. Rules 4 and 5 are vulnerable, though. This detector fails if the attacker performs the frontrunning and backrunning attacks from different addresses. Also, suppose the attacker splits either frontrunning or backrunning transactions in two. In that case, the profit calculation in rule 5 will wrongly calculate the profit (and the attack might seem unprofitable if the attacker splits the backrunning transaction).

B.3 EigenPhi

Another important player in the space of detecting MEV is EigenPhi.[6] They have a quite complex detection engine that looks for several types of MEV besides sandwich attacks. All the detection methods are rule-based. We summarize the heuristics they use in detecting sandwich attacks below. See their documentation for more details.[7]

1. They analyze every block and look first for a pair of transactions that might constitute a frontrunning and backrunning transaction. This is done by looking for any two transactions that are (1) sent to the same liquidity pool, (2) are in opposite directions, and (3) come from the same address.
2. Whenever such a pair of transactions is located, they analyze the transactions executed between them in the same block. If a transaction sent from a different address trades in the same direction as the frontrunning transaction, it is deemed a victim. It is unclear whether there can be multiple victims.
3. The profit is calculated by netting out the assets paid and received in the frontrunning attack with the assets from the backrunning attack.

As in the case for Flashbots in Appendix B.2, the EigenPhi method fails if the attacker uses different addresses for the frontrunning and backrunning attacks. Using additional addresses is simple and only requires simple balancing between the accounts in later blocks. Also, with this method, the profit estimate can be entirely wrong if the attacker splits one of the pieces of the sandwich in two.

B.4 Torres et al. [13] Sandwich Detection Heuristics

Torres et al. [13] presented a rigorous study of sandwich attacks. They rely on the same notions we defined at the start of this appendix, Appendix B, but use a different notation. They use r and s to denote the unique addresses assigned to users on the blockchain. more specifically, r_j denotes the $r$eciving address of a transaction sent by user j, and s_j denotes when the address of user j is registered as the sender of a transaction. The above notation summarizes their detection rules in the following 6 heuristics.

1. The frontrunning address and backrunning address must be equal ($r_{A_1} = s_{A_2}$).
2. The number of tokens bought in the frontrunning transaction must be within 1% of the tokens sold in the backrunning transaction.
3. All transactions must interact with the same pool (and be in the same block).
4. All sandwich transactions must have different transaction hashes.
5. The transaction indices of the frontrunning, victim, and backrunning transactions must be such that that was the realized order.
6. The gas prices must be in descending order of frontrunning, victim, and backrunning transaction.

As with most methods, this one also fails if the attacker uses different addresses for the frontrunning and backrunning transactions (rule 1). Furthermore, where Flashbots' and EigeinPhi's methods will misestimate profits if the attacker splits transactions, this method will fail to detect the attack (rule 2).

[6] https://eigenphi.io/

[7] https://eigenphi-1.gitbook.io/classroom/eigenphis-methodologies/how-eigenphi-identifies-mev/recognizing-cross-transaction-sandwich-mev, accessed August 21st, 2023.

B.5 Qin et al. [10] Sandwich Detection Heuristics

Qin et al. [10] also perform an insightful and rigorous analysis of sandwich attacks, but from the perspective of the victim, not the attacker (which is customary). Here, as well, we need to introduce some minor notation. Like us, they denote transactions as T_j, where transactions with $j =\in \{A1, A2\}$ belong to the attacker, and $j = V$ is the victim transaction. Where we denote the state of the liquidity pool as $\mathbf{X} = (X_1, X_2)$, they use $X = X_1$ and $Y = X_2$. The heuristics they used are summarized below.

1. Transactions for frontrunning T_{A1}, victim T_V, and backrunning T_{A2} must be in the same block and executed in that order.
2. Each frontrunning transaction T_{A1} is matched with exactly one backrunning transaction T_{A2}.
3. T_{A1} and T_V transact in the same direction, e.g., from X to Y, and T_{A2} transaction in the opposite direction, from Y to X.
4. The requirement of frontrunning and backrunning transactions coming from the same address is relaxed, and they only require T_{A1} and T_{A2} to be sent to the same smart contract address.
5. The assets sold in T_{A1} must be within $90\% \sim 110\%$ of the assets bought in T_{A2}, further relaxing the matching requirement.

Contrary to all other methods analyzed, this method places no requirement on the address of the frontrunning and backrunning transactions and is thus not prone to be fooled by this simple obfuscation tactic (rule 4). However, the method will fail to recognize sandwich attacks where the attacker splits either of their transactions such that the transaction volumes differ by at least 10% (rule 5).

C Additional motivating examples

In Section 4, we saw that extracting profits by constructing a sandwich attack is simple when only one user transaction is in the block B. This attack is, however, not permissible if the sequencer is subject to the verifiable sequencing rule put forth by Ferreira and Parkes [5] (defined in Definition 3 and pseudo code in Algorithm 1). In these simple cases, one can easily guarantee that no attack occurred by checking whether the 3 swaps in a block satisfy the sequencing rule, i.e., requiring a transaction order like (SELL, BUY, BUY) or (BUY, SELL, BUY).

The picture is more complicated in cases with more transactions in a block, $|B| > 1$. In the case where all user transactions are either all BUY or all SELL, the finding above still applies, as they can equivalently be considered pieces of one big transaction.

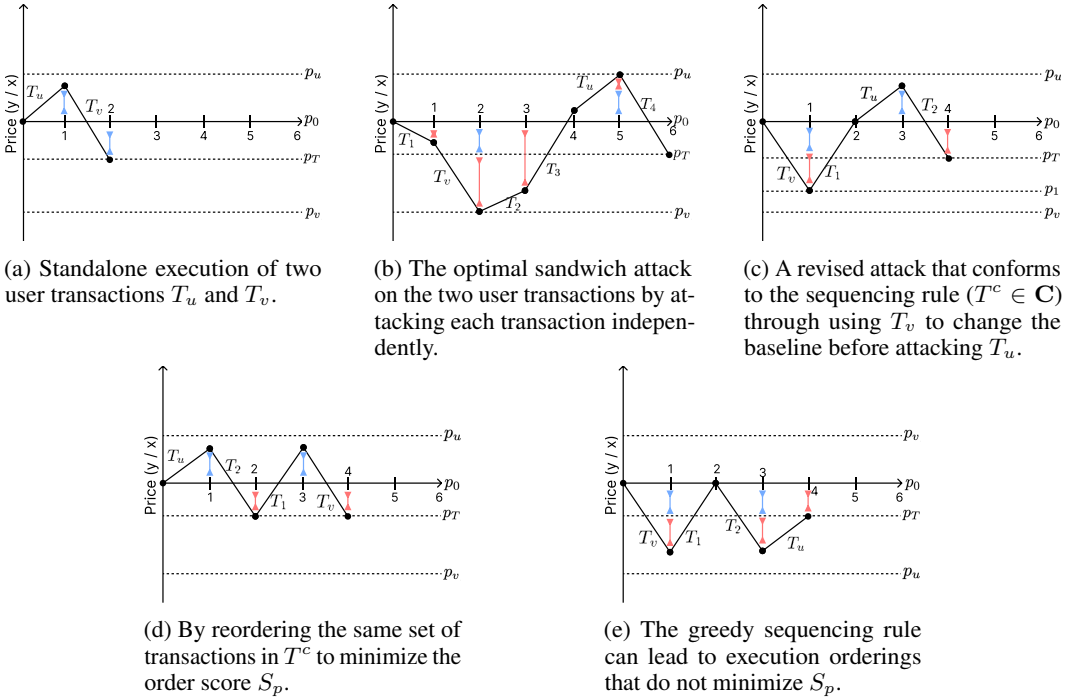
When user transactions are in opposite directions, values can be extracted while conforming to the sequencing rule, making detecting attacks more difficult. Let us consider an example where the original block has two user transactions $B^{(a)} = \{T_u, T_v\}$, with $T_u = \text{BUY}(q_u, p_u)$ and $T_v = \text{SELL}(q_v, p_v)$. The way to extract the maximal sandwich value given $B^{(a)}$ is to sandwich each user transaction separately, subject to their respective limit prices p_u and p_v, by inserting transactions $\{T_1, T_2, T_3, T_4\}$ into $B^{(a)}$, creating $B^{(b)}$, ordered as $T^{(b)} = (T_1, T_v, T_2, T_3, T_u, T_4)$, as shown in Figure 4b. *Note: In practice,* $T_2 = \text{BUY}(q_2)$ *and* $T_3 = \text{BUY}(q_3)$ *would be merged into* $T_{2,3} = \text{BUY}(q_2 + q_3)$.

Under the sequencing rule, however, $T^{(b)}$ would not have a valid transaction sequence. In this case, the attacker can let T_v execute first, moving the baseline price to a new, lower level p_1. From this new level, the attacker can now insert a frontrunning BUY transaction that exactly undoes T_v, $T_1 = \text{BUY}(q_u)$, then let T_u execute, and perform the backrun $T_2 = \text{SELL}(q_u)$ creating the order $T^{(c)} = (T_v, T_1, T_u, T_2)$, as shown in Figure 4c. Again, the attacker's profit (before fees) equals the difference between the price that user u would have paid if starting from p_1 and the one they did pay, starting from p_0. Interestingly, in this case, the optimal sandwich of T_u depends only on the volume of T_v, q_v, and not on q_u or p_u. This sequence of transactions has a decreased transaction order score than the maximal sandwich shown in Figure 4b, $S_p(T^{(c)}) < S_p(T^{(b)})$.

If we consider the same set of transactions as in Figure 4c, $B^{(c)} = \{T_u, T_v, T_1, T_2\}$, we can reorder them to create the order $T^{(d)}$ with a smaller order score $S_p(T^{(d)}) < S_p(T^{(c)}) < S_p(T^{(b)})$, shown in

Figure 4d. With this order of transactions, no actor can have made risk-free profits from a sandwich attack. This can be checked by partitioning the set of 4 transactions into two sets and checking for risk-free profits in any partition. This order also motivates the heuristic presented in Appendix D, as the important change between Figure 4d and Figure 4e is that the smallest transaction is first in the former, but not in the latter.

Until now, a reduced order score S_p has meant less sandwich MEV extracted. However, by considering Figure 4e, it becomes clear that a higher score does not necessarily imply the presence of an attack, as no profitable sandwich exists in this block, $B^{(e)}$.



(a) Standalone execution of two user transactions T_u and T_v.

(b) The optimal sandwich attack on the two user transactions by attacking each transaction independently.

(c) A revised attack that conforms to the sequencing rule ($T^c \in \mathbf{C}$) through using T_v to change the baseline before attacking T_u.

(d) By reordering the same set of transactions in T^c to minimize the order score S_p.

(e) The greedy sequencing rule can lead to execution orderings that do not minimize S_p.

Figure 4: When there are 2 user transactions T_u and T_v (shown in the first plot), the problem of detecting attacks. The second plot shows how much the deviation from the baseline increases when a sandwich attack is inserted. With two user transactions, it is possible to create valid sandwich attacks under the greedy sequencing rule. Notice that there is a successful attack in (c) but not in (e), but the score for the order in (c) $S_p(T^c)$ is larger than the order in (e) $S_p(T^e)$.

D Heuristic Approximation to the Optimal Execution Order

In this section, we introduce the polynomial-time algorithm for approximating the optimal transaction order T^*. Before we present the algorithm, we start by introducing a concept crucial for constructing valid transaction sequences, a *verifiable sequencing rule* Ferreira and Parkes [5].

Definition 2 (Verifiable Sequencing Rule)**.** *A sequencing rule, where all valid sequences under which is defined as the set* $\mathbf{C}$*, is verifiable if there is there exists a polynomial time algorithm to check for any block order* T *if* $T \in \mathbf{C}$*.*

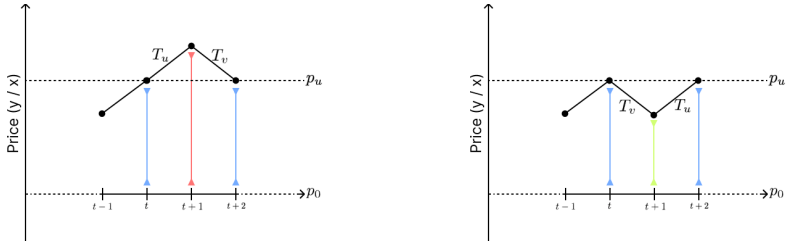
Ferreira and Parkes [5] also presents a greedy algorithm that is an efficient instance of a verifiable sequencing rule, called a *greedy sequencing rule*, defined below. The complete algorithm is given in Algorithm 1 in Appendix D.

Definition 3 (Greedy Sequencing Rule)**.** *In three steps, the greedy sequencing rule constructs valid and verifiable sequences* $T \in \mathbf{C}$*. (1) split the set of transactions into the set of buys and the set of sells,* $B = Q^{Buy} \cup Q^{Sell}$*. (2) As long as* $|Q^{Buy}| > 0 \wedge |Q^{Sell}| > 0$*, execute any remaining sell order if current price* $p_t \geq p_0$*, otherwise execute any buy. (3) When only buy or sell orders remain, execute them in any order.*

Since the set of transaction orders produced by the greedy sequencing rule is a subset of all orders, we use the following theorem to restrict the search space from B_n to $\mathbf{C}$.

Theorem 1. *Any optimal* $T^* \in \underset{T \in B_n}{\text{argmin}}\, S(T)$ *is also a valid ordering under the greedy sequencing rule, i.e.,* $T^* \in \mathbf{C}$.

Proof. See Figure 5 for visual intuition. We show this by contradiction. Assume an optimal T^* is not contained in $\mathbf{C}$, i.e., $\exists T \in B_n$ s.t. $T \in \text{argmin}_{T \in B_n} \wedge T \notin \mathbf{C}$. From $T \notin \mathbf{C}$, we know that at least one set of two consecutive buy orders must execute at a price $> p_0$ followed by a sell order. Denote the sell order T_v and the buy order immediately preceding it T_u, shown in Figure 5a. From here, we can swap the order of T_u, T_v to construct an alternate order T', shown in Figure 5b. Now, it is easy to see that $S(T) > S(T')$ since the prices before and after T_u, T_v executes, P_t, P_{t+2}, are the same regardless of the order, while the price between T_u, T_v executes, P_{t+1}, is strictly smaller. But since $S(T')$ is smaller than for the assumed optimal T, T cannot be optimal, showing that our original assumption is wrong, i.e., $T^* \in \mathbf{C}$. □



(a) The assumed optimal ordering T.

(b) Alternative ordering T' that complies with sequencing rule is strictly better than the original.

Figure 5: Visual intuition for why any optimal transaction execution order must conform to the verifiable sequencing rule.

Ferreira and Parkes [5] prove that when the greedy sequencing rule is applied, a large class of exploitative attacks are no longer feasible, providing a solid level of user protection. We also empirically see reduced deviations from baseline under the sequencing rule, providing more robust evidence for the validity of our approach.

D.1 Volume Heuristic Greedy Sequencing Rule

In Algorithm 1, we present our version of the greedy sequencing rule put forth by Ferreira and Parkes [5]. The main algorithm is the same, and the main addition is that where they allow any valid transaction to execute in any order, we always complete the smallest transaction among the valid ones. Looking at an example sequence in Figure 6 might be helpful to understand the algorithm.

Before the algorithm executes, it needs a vector $\mathbf{X}_0$ that defines the initial state of the exchange, in terms of the currently locked up token reserves $(X_{0,1}, X_{0,2})$ at time $t = 0$. In the algorithm, we represent the state by the current price in the exchange $x_t \frac{X_{t,2}}{X_{t,1}}$. We also have an unordered set of transactions $B = \{T_1, ...T_n\}$ that are available for execution that the sequencer might have picked up from the public peer-to-peer network, received directly from MEV searchers through bribes [14], or they created themselves. Each transaction $T_i = \text{Buy}(q_i, p_i)$ (or Sell) defines the number of tokens q_i to trade and a limit price p_i for the trade. The output of the algorithm will be an ordered sequence of the n transactions in B, $T = (T_{\sigma_1}, ..., T_{\sigma_n})$, with the permutation operator σ_i defined by the algorithm.

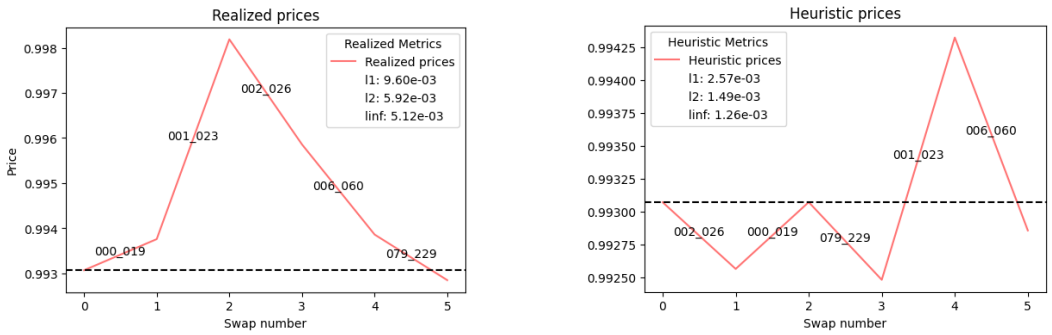
The algorithm partitions the set of transactions B into the subset of buy orders Q^{buy} and the subset of sell orders Q^{Sell} before processing any transactions. The basic greedy sequencing rule by Ferreira and Parkes [5] does not restrict how the transactions in each of these queues are ordered. In this algorithm, though, we sort by the transaction volume q_i in ascending order. Then, as long as both buy and sell orders are left, we execute the next buy order if the current price x_t is below the initial price

x_0, or the next sell order otherwise. When there are only buy transactions or only sell orders, they are all executed according to the order regardless of the current state x_t. We analyze the performance of the approximation in Appendix D.2.

Algorithm 1 Volume Heuristic Approximation to T^*

Input: Initial pool state $\mathbf{X_0}$; Set of transactions B
Output: Execution ordering T s.t. $S(T) \gtrapprox S(T^*)$

Initialize T to an empty list
Partition B into the subset of buy orders $Q^{\text{buy}} \subseteq B$ and sell orders $Q^{\text{sell}} \subseteq B$
Order Q^{buy} and Q^{sell} according to transaction volume ascending
while $|Q^{\text{sell}}| > 1$ and $|Q^{\text{buy}}| > 1$ **do**
    $t \leftarrow |T|$
    if $x_t = x_0$ **then**
        $A \leftarrow$ pop off the $A \in \{Q_1^{\text{buy}}, Q_1^{\text{sell}}\}$ that minimizes $|x_{t+1} - x_t|$
    else if $x_t \leq x_0$ **then**
        $A \leftarrow$ pop off the smallest buy order Q_1^{buy}
    else
        $A \leftarrow$ pop off the smallest sell order Q_1^{sell}
    end if
    Append A to the end of T
    $x_{t+1} \leftarrow$ the result of executing A
end while
If $B^{\text{buy}} \cup B^{\text{sell}}$ is non-empty, append all $A \in B^{\text{buy}} \cup B^{\text{sell}}$ to T



(a) The observed transaction execution order in a realized block B.

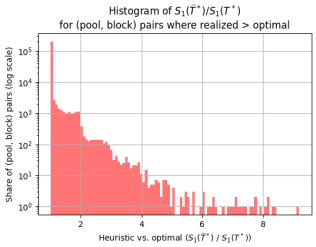
(b) A counterfactual order of B ordered according to the volume-based heuristic in Algorithm 1.

Figure 6: This example shows how the realized block price trajectory can be effectively changed to something with smaller $S_p(T)$. Interestingly, the left figure contains an order that could constitute a sandwich attack, while it is entirely removed in the order produced by the heuristic.

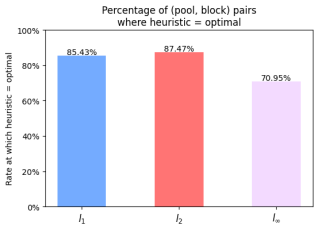
D.2 Empirical Comparison of Heuristic $\widehat{T}*$ to T^*

To empirically validate that the approximation to the optimal execution order T^* produced by Algorithm 1, we calculate the true optimal T^* for each of the p-norms S_1, S_2, S_∞ for blocks small enough to allow for enumeration, $|B| < 8$. *Note: This analysis is performed on a subset of the data for which the realized order T has a different score than the optimal T^*, i.e., $S_1(T) \neq S_1(T^*)$ to weed out possibly trivial cases.* Figure 7a shows the distribution of values of $S_1(T)$ for the heuristic ordering divided by the true $S_1(T^*)$ on a log scale. This makes it apparent that the heuristic agrees exactly with the optimal in most cases, even for the 'hard' subset of pool-block pairs. Figure 7b shows the percentage of blocks where the heuristic exactly equals the true optimal for the subset of the data. The heuristic agrees with the optimal in 71% to 87% of the pool-block pairs measured. Figure 7c shows what the mean deviation from the optimal score is, i.e., we calculate $\frac{1}{N}\sum_B \frac{S_p(T')}{S_p(T^*)}$ for each

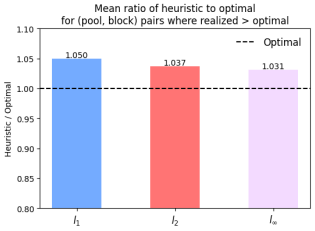
block B for which $S_1(T) \neq S_1(T^*)$. The mean deviation ranges from 1.031 to 1.050. Considering that these numbers are for the subset of transactions that are not trivial, the numbers for the entire dataset would be much closer to optimal. This makes us conclude that this approximation is serving its purpose.



(a) Histogram for the distribution of the heuristic order score divided by the true optimal score $\frac{S_1(T')}{S_1(T^*)}$ for $|B| < 8$



(b) When removing cases where the original order was not optimal, the heuristic order T' is optimal in 85%, 87%, and 71% of cases for the S_1, S_2, and S_∞ scoring functions, respectively.



(c) The mean ratio between the score S_p for the heuristic order T' to the true optimal T^* whenever the realized is not optimal is 1.05, 1.04, and 1.03 for S_1, S_2, and S_∞, respectively.

Figure 7: Comparisons of the approximation to the optimal execution ordering T^* achieved with Algorithm 1 of the true value of T^* in those cases where it is feasible to calculate it by enumeration ($|B| < 8$).

E Dataset and Data Collection

E.1 Data Description

To perform the empirical analysis, we collect a dataset of all Uniswap V3 swaps written to the Ethereum blockchain since the launch of the Proposer-Builder Separation (PBS) implementation MEV-boost. The dataset encompasses data from 1,866,537 blocks, from block 15,537,940, written September 15, 2022, to block 17,864,015, written August 7, 2023. There are 7,994,736 swaps in 6,292 different pools, and we analyze 6,864,029 block-pool pairs, i.e., the number of blocks with swaps for each pool. The dataset includes a binary variable indicating whether the block was proposed through the MEV-boost auction mechanism (i.e., the vanilla block-proposing mechanism). About 87% of the blocks in the dataset were proposed through the MEV-boost auction. All data used will be publicly available for reproducibility and further analysis.

E.2 Data Collection System

To collect the data we used in the analyses in this work, we had to prepare some infrastructure. See Figure 8 for a schematic overview of the system (made using Lucid Chart software[8]). The system is built around the Ethereum full-node (marked A in Figure 8). This node is connected to the wider Peer-to-Peer (P2P) Ethereum network, where it will contribute to securing the blockchain by validating new blocks and rebroadcasting new candidate transactions in the *public mempool*. The public mempool is volatile storage where transactions are held from the time a user sends it to the network until the time it is permanently to the blockchain. Since it is volatile, i.e., there is no permanence, we must continuously monitor it and store the members as they come in. This storage we achieved by having a PostgreSQL relational database (B) and a MongoDB document database (C) as a backup.

[8] https://lucid.app/

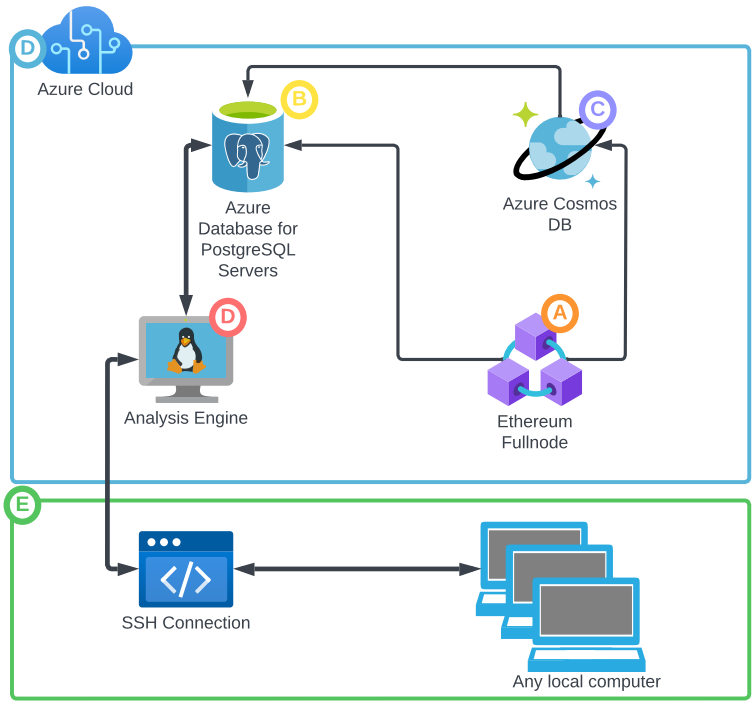


Figure 8: Schematic overview of the data collection system we built in June of 2023 to continuously collect data for analysis. All the important heavy lifting is happening in the Microsoft Azure cloud, and the analysis computer exposes an SSH endpoint one can connect to from one's local computer to interact with the system.

E.3 Descriptive Analysis

In Figure 9, we show the distribution of how many transactions there are in each block within each trading pool for all blocks in our dataset. The left plot clearly shows that most blocks contain only 1 transaction. The right subplot shows the distribution on a log scale (y-axis) to emphasize the tail of the distribution as well. Interestingly, all extractive attacks happen in the subset of blocks with 2 or more transactions (the tail).

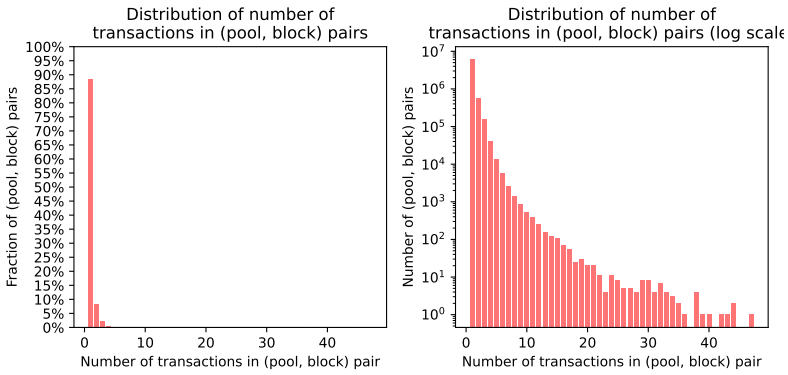


Figure 9: Distribution of the number of transactions in each block across all blocks in our dataset. Left: Frequency in percentage of all blocks (y-axis) for each number of transactions per block (x-axis). Right: the same distribution is shown in absolute numbers with a log scaling of the y-axis.

As prior work has shown (see Appendix A), the MEV-boost auction mechanism has gained much traction since its release in 2021. In Figure 10, we show the distribution of the share of books built through each mechanism. In our data set of almost 2 million blocks, only 14% transactions are built in the standard manner laid out by the Ethereum protocol [6], while the rest is built through the auction mechanism where actors pay for the right to build blocks. This hints at the prevalence of profitable attacks on the Ethereum blockchain currently.

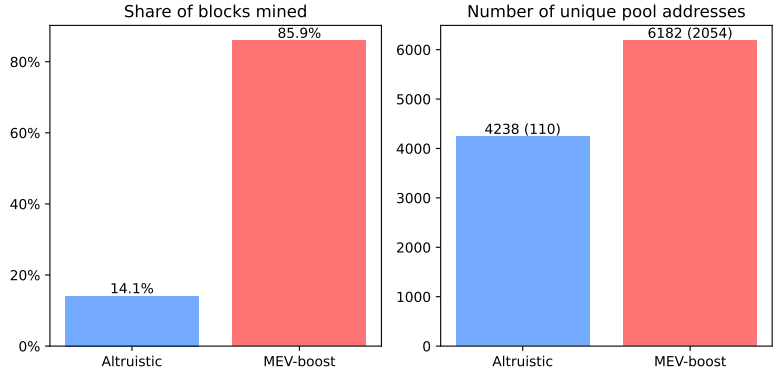


Figure 10: Distribution of blocks built through each of the mechanisms we consider in this paper. The blue bar is the share of blocks built through the standard method described in the Ethereum protocol, while the red bar is the share of blocks built by actors who paid for the right to build the block in an MEV-boost auction.

F Further Analysis of Empirical Data

In this section, we present more ways to slice the data we present in Appendix 5 for the interested reader. In Figure 11, we present a comparison between the number of transactions the average block has depending on whether that block was built in the standard manner defined by the Ethereum protocol, i.e., a *vanilla* block or through the MEV-boost auction mechanism. Since most blocks have only 1 transaction per pool, the groups look similar on average. However, among the subset of blocks with 2 or more transactions, interesting differences arise (middle and right plot).

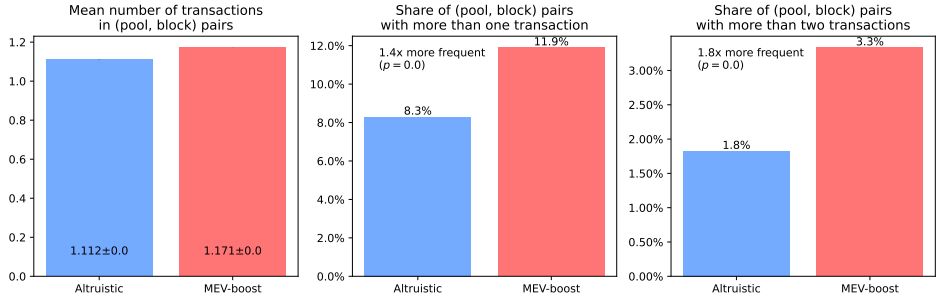
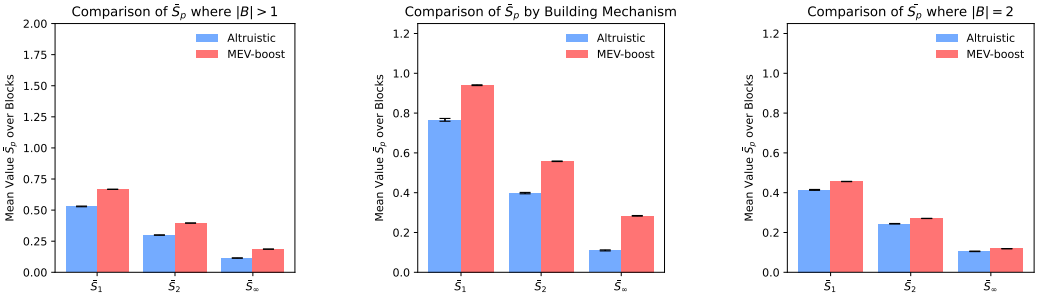


Figure 11: When we look at the subset of transactions written to the blockchain through the MEV-boost action mechanism vs. the standard protocol mechanism (called vanilla), we see that the mean number of transactions in a block per pool is very similar (left). When we consider the subset of blocks with more than 1 transaction, the picture changes, and we see that 12% of MEV-boost blocks have 2 or more transactions vs. 8% for vanilla (middle). For blocks with 3 or more transactions, this difference becomes even larger, and almost double the share of MEV-boost blocks are in this category compared to vanilla blocks (right).

Figure 12 shows the same picture as in Figure 3 in Appendix 5, but also includes the bigger subset of blocks that have 2 or more transactions in them (left) in addition to the subset with 3 or more

(right). The effect we observe is the same in both cases, only more pronounced when considering only $|B| > 2$. However, when we look at the subset of blocks that contain exactly 2 transactions, $|B| = 2$, the effect largely vanishes, as shown in Figure 12c. This indicates that the measure D_p specifically picks up on attacks like the sandwich attack that requires at least 3 transactions in a block.



(a) For blocks with 2 or more swaps, the measure D_p increased between mechanism blocks and those created through the MEV-boost auction.

(b) For the subset of blocks with 3 or more swaps, the difference of the measure D_p increases several times.

(c) For the subset of blocks with exactly 2 transactions, the difference of the measure D_p largely disappears.

Figure 12: The measure D_p is meaningfully larger for blocks built through the MEV-boost action than vanilla blocks. Here, we only focus on the non-trivial blocks with more than 1 swap and see a bigger difference for blocks with 3 or more swaps.

In Figure 12, we saw that when we look at the entire dataset, we can distinguish the altruistic group from the malicious group with very high confidence. What happens if we want to classify a single new sequencer? In Figure 13, we look at the average metric estimate and the standard error for an increasing number of observations $|\mathcal{D}|$. The experiment relies on random sampling of subsets of the data, so we ran the procedure 50 times to make the results more reliable. What we observe is that we can distinguish malicious and altruistic sequencers at a 58% confidence interval after around 50 blocks ($|\mathcal{D}| \geq 50$), and at a 95% confidence interval, we need around 200 blocks observed ($|\mathcal{D}| \geq 200$). Seeing that there is a new block produced every 12 seconds, and we have hundreds (and thousands for some) for many sequencers, these numbers indicate that one can pick out malicious actors with longitudinal observation.

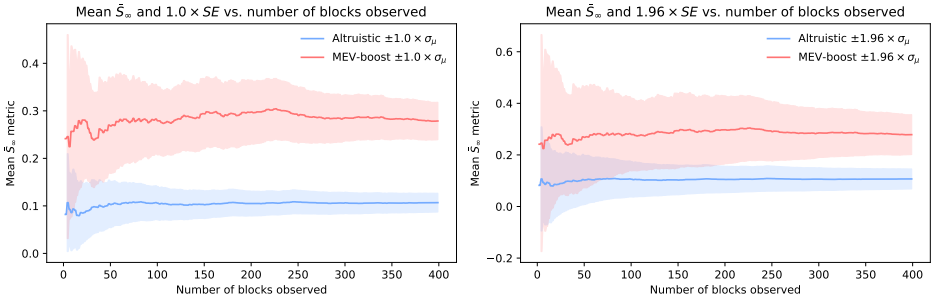
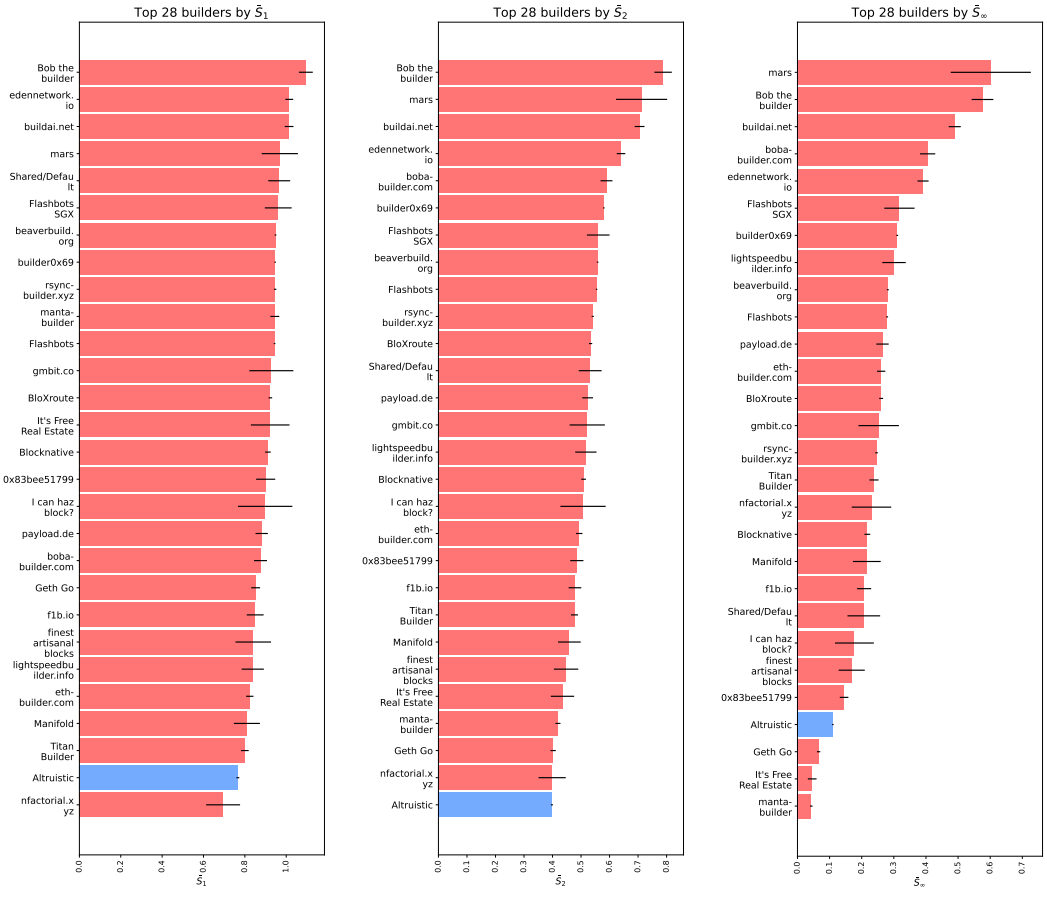


Figure 13: Standard error of estimation of the within-group value of the surveillance metric $\bar{S}_p$ with an increasing number of blocks observed, i.e. $|\mathcal{D}_b|$. (a) At a 1 standard error level (68% confidence interval), the metric estimates seize overlapping after 50 blocks have been observed. (b) For the 95% confidence interval, we need $|\mathcal{D}| \geq 200$ to determine whether the source is adversarial or not.

Lastly, we consider what happens when we apply the surveillance metric for individual builders and relays, Figure 14 and Figure 15. The results are fairly intuitive in that most participants in the MEV-boost auctions have significantly higher values for the metric than the altruistic sequencer. However, a couple of things are of note. First, there is a large range of surveillance metric values

between the builders and relays, indicating that some builders are much more active/successful in extracting MEV. We want to investigate this assumption more closely in the future. Second, among the `bloxroute` relays, the one advertised as being 'ethical' also scores the lowest on the surveillance metric. The score is, however, higher than the altruistic baseline.

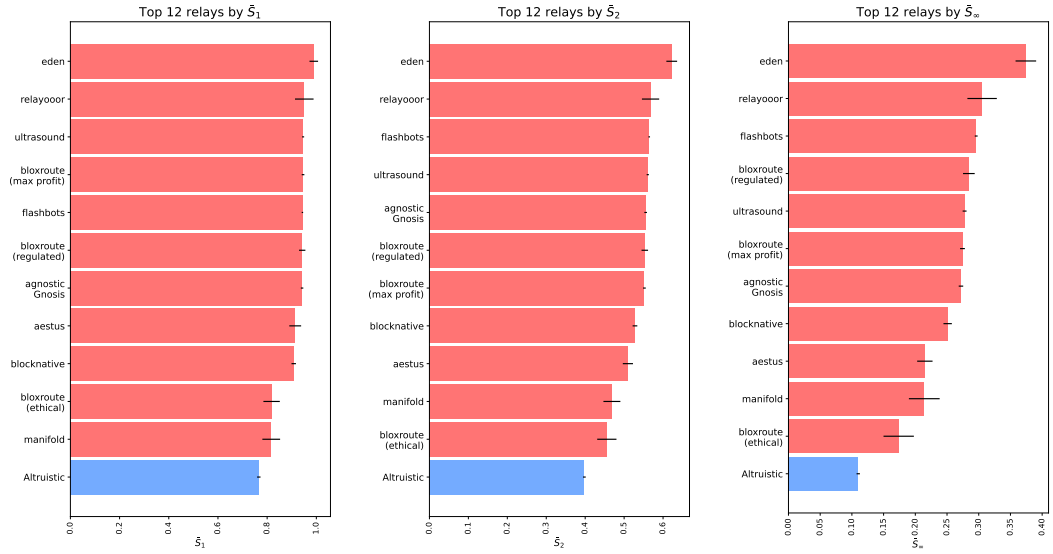


(a) Top builders ordered by their mean surveillance metric $\bar{S}_1$.

(b) Top builders ordered by their mean surveillance metric $\bar{S}_2$.

(c) Top builders ordered by their mean surveillance metric $\bar{S}_\infty$.

Figure 14: We see significantly higher levels of the surveillance metric $\bar{S}_p$ for almost all builders that participate in the MEV-boost auction compared to the altruistic. One notable exception is `Geth Go`, which has the lowest $\bar{S}_\infty$ metric. This finding stands to reason since this is presumably the standard Ethereum node implementation that does not implement any MEV searching.



(a) Top relays ordered by their mean surveillance metric $\bar{S}_1$.

(b) Top relays ordered by their mean surveillance metric $\bar{S}_2$.

(c) Top relays ordered by their mean surveillance metric $\bar{S}_\infty$.

Figure 15: An overview of the top relays according to their corresponding surveillance metric $\bar{S}_p$. All relays who participate in the MEV-boost auction have significantly higher levels than the altruistic relay.

G Tests of Significance in Empirical Results

We perform a standard student's t test on the mean value of the metrics between the MEV-boost and vanilla groups. The mean of the metric for the p-norm for the subset of swaps in group $g \in \{\text{Vanilla}, \text{MEV-boost}\}$ is denoted by $\overline{D_p(\mathbf{T^g})} = \frac{1}{|\mathbf{T}^g|}\sum_{T\in\mathbf{T}^g} D_p(T)$, where $\mathbf{T}^g$ is the subset of all observed block execution sequences where the block belongs to group g, i.e., $\mathbf{T}^g = \{T | T \in g\}$. The null hypothesis is $H_0 : \overline{D_p(\mathbf{T}^{\text{Vanilla}})} = \overline{D_p(\mathbf{T}^{\text{MEV-boost}})}$. By the student's t-test, we get $p < 10^{-9}$, i.e., a strong statistical significance.

In particular, the test we apply in this case is the more general statistical test called Welch's t-test, which adjusts for cases where there is an unequal sample size and variance in the two populations [11, 3].

Welch's t-test define the t-statistic as [12],

$$t = \frac{\Delta\bar{X}}{s_{\Delta\bar{X}}} = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{s^2_{\bar{X}_1} + s^2_{\bar{X}_2}}} \tag{1}$$

$$s_{\bar{X}_i} = \frac{s_i}{\sqrt{N_i}}, \tag{2}$$

where $\bar{X}_i$ is the sample mean, $s_{\bar{X}_i}$ the corresponding standard error, with s_i being the corrected sample standard deviation, and N_i the sample size of group i. This test statistic t will also be approximately distributed according to the t distribution with ν degrees of freedom. We refer to Allwood [1] for the exact formula for ν, as it will not further the narrative to include it herein. The test concludes with constructing the cumulative density function and calculating a probability p that we see any $t' \sim t_\nu$ exceeding our test statistic t in absolute value $p = P_{t'\sim t_\nu}(|t'| > t \mid \mathcal{D})$ (i.e., we condition on the observed data).