# Frontrunning Attacks on Decentralized Exchanges: Batch Auctions as a Market Design Response

Max Resnick[1]

[1]Department of Economics, MIT

**Abstract**

Decentralized exchanges facilitate billions of dollars in order flow on chain every day. But the subtleties of order execution allow traders to be exploited by sandwhich attacks. These attacks in the first order are simply a transfer from traders to extractors so they don't necessarily cause welfare loss; however, once traders adjust their behavior to guard against attacks, orders sometimes fail, the mempool becomes congested, and the market for blockspace unravels, increasing the expected time to transact for everyone. I present a simple model of a frontrunning game and show that it leads to failed transactions, that would not fail without the threat of frontrunning. Then I present a simple batch auction mechanism that completely eliminates frontrunning.

## 1 Introduction

Blockchains are distributed append only ledgers but the order in which transactions are appended isn't first come first serve. Users who wish to have transactions included in blocks first create a message containing the transaction they wish to have included and then produce a signature using their private key. They then send this message, along with the valid signature into the memory pool (mempool) which holds pending transactions before they are added to the chain via their inclusion in a valid block. In order to incentivise miners to include their transaction, users can add a miner tip to their transaction which is a transfer from the sender to whoever mines the block that is valid if and only if the transaction is included within a block. Often miner's assemble transactions within the block ordered by this priority fee (although in some cases this is not a requirement). It is this mapping from miner tips to height within a block which conceals enormous complexity and creates inefficiencies which have begun to cause the market for block-space on Ethereum and other chains to unravel.

The problem with a direct mapping from fees to blockheight is that it allows bots to snipe transactions in the mempool and perform value extracting manipulations such as sandwhich attacks, stealing arbitrage opportunities, frontrunning liquidations, and more. Collectively, these actions are referred to as Miner Extracted Value (MEV) although the agents who engage in these activities are not always miners.

The possibility of MEV causes users to avoid the mempool altogether in favor of private mining pools and other mechanisms which means that their transactions take much longer to mine. One such dark pool is the flash-bots auction which bundles transactions in a fixed order to avoid the risk of MEV and transaction failure.

## 2 Related Work

The inspiration for this paper is [1] which is an argument for frequent batch auctions in traditional markets where the analouge of MEV is high frequency trading.

Most existing MEV discussion is still focused on the consensus level [2] [3] [4]. The sparse literature that has explored this topic mainly focuses on identifying the problem, sometimes incorrectly, and often still has the feel of a distributed systems paper rather than a market design paper [5]. The closest paper I found to this topic was [6] which correctly identified the interplay between slippage tolerance and front running profitability but did not propose any solutions.

The literature on Automated Market Makers, as they exist on chain is quite robust in contrast. See [7] and it's references. A cartel of convex analysts have staked out the position that "Convex Analysis seems to be the right way to look at AMMs" as a member of this group recently recounted to me. I tend to agree with this opinion, which is why I resort to their definition of a CFMM throughout.

### 2.1 Flashbots

Flashbots is the largest dark pool for Ethereum blockspace. The Flashbots auction is a first price sealed bid mechanism in which trader's submit bids along with preferences for the order of their transaction within the block. Transactions are also precompiled so they cannot fail on chain. Flashbots is more attractive to DEX users because it provides frontrunning protection; however, it's not a perfect solution. In fact its more of a symptom of the problem. The problem with flashbots is that it fragments the mining power of the network which increases the expected order execution time for everyone (including those using flashbots) [8].

## 3 Brief Description of Automated Market Makers

Automated Market Makers were first proposed in the context of prediction markets [9]. They started gaining real traction as a mechanism when people realized that they could be implemented as a smart contract on a blockchain. From this realization, the first decentralized exchanges were born starting with Uniswap v1 [9]. Over the years, these mechanisms evolved and today they are responsible for facilitating a significant share of crypto market volume. Every successful AMM has been part of a class of Constant Function Market Makers (CFMMs).

**Definition 3.1** (Constant Function Market Maker (CFMM))**.** A constant function market maker is described by a trading function:

$$\varphi : \mathbb{R}^n_+ \to \mathbb{R} \tag{1}$$

Here $R \in \mathbb{R}^n$ is the reserve vector denoting how much of each asset the market maker has on hand. A proposed trade $(\Delta, \Lambda)$ is a tuple containing a provision basket $\Delta$ of assets that the trader adds to the pool and a withdrawl parameter $\Lambda$ containing assets the trader will recieve. In keeping with the name, the market maker accepts any trade $(\Delta, \Lambda)$ provided that it's trading function $\varphi$ remains constant:

$$\varphi(R + \Delta - \Lambda) = \varphi(R) \tag{2}$$

**Properties.** We will also assume that $\varphi$ is concave, increasing, and differentiable [7].

**Example 3.1.** The most widely used CFMM is the constant product market maker which is implemented by the trading function

$$\varphi : \mathbb{R}^2_+ \to \mathbb{R} \tag{3}$$

defined by

$$\varphi(R_1, R_2) = R_1 R_2 \tag{4}$$

If a trader adds $q_1$ of token 1, we can derive the amount of token 2, $q_2$ that they recieve as

$$\varphi(R_1, R_2) = \varphi(R_1 + q_1, R_2 - q_2) \tag{5}$$

$$R_1 R_2 = (R_1 + q_1)(R_2 - q_2) \tag{6}$$

$$q_2 = \frac{R_2 q_1}{R_1 + q_1} \tag{7}$$

## 4 The Memory Pool and Miner Extracted Value

When a trader wishes to place an order in an on chain market, they come up with a list of smart contract function calls known as a transaction then cryptographically sign it with their private key. The signed transaction is then transmitted to a few known full nodes who verify that the transaction is valid and that the signature corresponds to the sender's public key. If it passes these checks, the transaction is added to the full node's memory pool and the transaction is passed on to other nodes on the network. The memory pool is where a full node stores valid transactions before they are added to the chain. When the node receives news that a new block has been mined, it reaches into the memory pool to grab enough of these transactions to fill a block and then repeatedly hashes this list in an attempt to mine it.

### 4.1 Sandwhich Attacks

In principle, miners on Ethereum have unlimited flexibility in how they order transactions within blocks. This allows them to assemble transactions in such a way as to extract the maximum possible value. We will focus on sandwhich attacks.

The attacker observes a particularly juicy beef patty (transaction) in the mempool he adds a top bun (an order in the same direction as the trader) just above the transaction and a bottom bun (an order in the opposite direction of the same magnitude as the top bun). If executed by the miner, this strategy can produce risk free profit.

**Example 4.1.** Suppose we have the constant product AMM defined in the previous example. Let $R_1 = R_2 = 100$. Suppose a trader arrives at the market wishing to trade $q_1 = 1$ of token 1. Plugging into our formula from the previous example, the trader receives:

$$\frac{100 \cdot 1}{100+1} = \frac{100}{101} \approx .99 \tag{8}$$

Now consider what happens when a miner sandwiches with an order of size 1. The miner's 1 unit buy order executes first, bringing the new reserves to

$$\left(100+1, 100 - \frac{100}{101}\right) \tag{9}$$

Now the trader executes his 1 unit buy order and receives

$$\frac{100 - \frac{100}{101}) \cdot 1}{(100+1)+1} = \frac{100 - \frac{100}{101}}{102} \approx .97 \tag{10}$$

Bringing the reserves to

$$\left(100+1+1, 100 - \frac{100}{101} - \frac{100 - \frac{100}{101}}{102}\right) \approx (102, 98.04) \tag{11}$$

Finally, the miner's sell order of magnitude equal to the amount of token 2 he gained in the first transaction executes. Since he is trading the other way, he receives

$$\frac{R_1 q_2}{R_2 + q_2} = \frac{102 \cdot .99}{98.04 + .99} \approx 1.02 \tag{12}$$

To recap, the miner put in 1 unit of token 1 and got out 1.02 so he makes a risk free profit of .02 of token 1. Notice that this aligns with how much the trader lost from being frontrun. This is not a coincidence.

To generalize this example, we need one more definition.

**Definition 4.1.** Given a CFMM defined by $\varphi : \mathbb{R}_+^2 \to \mathbb{R}$, let $\psi : \mathbb{R}_+ \times \mathbb{R}_+^2 \to \mathbb{R}_+$ be the parameterized solution to the invariant:

$$\varphi(R - x + \psi(x)) = \varphi(R), \qquad \forall x \in \mathbb{R}_+ \tag{13}$$

That is, if a trader tenders $x$ of token 1, $\psi(x, R_1, R_2)$ is the amount of token 2 the trader receives.

**Example 4.2.** In the constant product case, we had :

$$\varphi(R_1, R_2) = R_1 R_2 \tag{14}$$

We derived $\psi$ as

$$\psi(q_1, R_1, R_2) = \frac{R_2 q_1}{R_1 + q_1} \tag{15}$$

Armed with this definition, we can see that the value extracted by frontrunning an order of size $x$ with an order of size $y$ on a CFMM with parameterized trading function $\psi$ is

$$\psi(x, R_1, R_2) - \psi(x, R_1 + y, R_2 - \psi(y, R_1, R_2)) \tag{16}$$

Notice that this is just the difference between what the trader would receive if they were not sandwiched and what they actually receive since the game is zero sum.

## 4.2 Defence Against Front-running

Notice that since the trading function $\varphi$ is concave, a miner who could frontrun with an order of arbitrary magnitude could harvest the entire order size of the original trader. Of course this cannot happen in practice otherwise there wouldn't be billions of dollars trading on decentralized exchanges. The first barrier is fees. In practice, AMMs charge a small percentage fee $\gamma \in (0, 1]$ for each trade. which prevents miners from say submitting on order of size 200 to frontrun a transaction of size 1 because they will end up paying too much in fees to make up the losses. In practice fees tend not to be the limiting factor and they significantly complicate the mathematics so we ignore them for the remainder of this paper. The second issue is blockspace. A sandwhich attack consumes two blockslots, one for the frontrun and one for the return. Even if the miner is the one doing the frontrunning, he still has to pay the opportunity cost of the miner tip that he would have received if he included two transactions from the mempool rather than the two attacking transactions. The third issue is slippage tolerance. When traders submit a buy order, they also submit a slippage tolerance $s$ which is the maximum price at which their order will execute. If the state of the AMM is at a point where the price is beyond this threshold by the time it reaches the transaction, the transaction will revert and no trade will take place. For ease of notation, it is useful to convert this parameter $s$ into a restriction on the amount of reserves of token 1 that the trader will tolerate when their transaction executes, which I will refer to as $S$. An expressive AMM, one which can express every price ratio $p \in (0, \infty)$, permits a one to one and onto mapping between restrictions on prices $s$, and restrictions on reserves $S$ so the substitution is purely for convenience in most cases.

### 4.3 Optimal Sandwich Attacks

Given an order of size $x$ with tolerance $S$ and miner tip $T$, reserves of $(R_1, R_2)$ and a parameterized trading function $\psi$, what is optimal behavior for the miner? Without considering fees, if the miner does sandwich, he will submit as large a frontrunning order as possible without pushing the reserves over the threshold $S$. So his order size will be $S - R_1$. His extracted value will be

$$\psi(x, R_1, R_2) - \psi(x, S, R_2 - \psi(S - R_1, R_1, R_2)) \tag{17}$$

Accounting for the opportunity cost $2T$, this is profitable if

$$\psi(x, R_1, R_2) - \psi(x, S, R_2 - \psi(S - R_1, R_1, R_2)) \geq 2T \tag{18}$$

The threshold value of $R_1$ at which the above inequality holds with equality, $R^*$, can be solved for generally by using a bracketing zero finder of

$$\psi(x, R_1, R_2) - \psi(x, S, R_2 - \psi(S - R_1, R_1, R_2)) - 2T = 0 \tag{19}$$

## 5 Model

Let us initially focus on a single trader. The trader arrives at the market with an urgent inelastic need to buy 1 unit of token 1 on an AMM with trading function $\psi$. The trader knows the cumulative distribution $F(\cdot, T)$ of the change in the reserves of token 1 that he will observe for each miner tip $T$ that he may submit. Assume for simplicity that the random variable described by $F(\cdot, T)$ is symmetric about the initial reserve of token 1, $\overline{R}_1$. If $T > T'$, we will assume that $F(|a - \overline{R}_1|, T) > F(|a - \overline{R}_1|, T')$. Intuitively, this is a strong version of the condition that a higher position within the block means that the price movement before the order executes has lower variance.

The trader submits an order which includes two parameters: slippage tolerance $S$, and miner fee $T$. He then earns $-T$ if the order does not execute at all, $\psi(1, S, R_2 - \psi(S - R_1, R_1, R_2)) - T$ if the order executes but is frontrun, and $\psi(1, R_1, R_2)$ if the order executes and is not frontrun. Let $R^*(S)$ be the threshold reserve at which it becomes profitable to frontrun the transaction if the realized reserve ends up below that amount. Then the expected utility for the trader who submits $(S, T)$ is

$$E(u(S,T)) = F(R^*(S,T),T) \cdot \psi(1,S) + \int_{R^*(S,T)}^{S} f(x,T)\psi(1,R_1+x)dx \tag{20}$$

Note that we have dropped the third parameter in $\psi$ since it can be parameterized by the second parameter. Holding $T$ fixed, we compute the optimal $S$, $S^*(T)$ as the solution to the first order condition:

$$\frac{\partial R^*}{\partial S}(S,T)f(R^*(S,T),T)\psi(1,S) + F(R^*(S,T),T)\frac{\partial \psi}{\partial S}(1,S) \tag{21}$$

$$+ \frac{\partial R^*}{\partial S}(S,T)f(R^*(S,T),T)\psi(1,R_1 - R^*(S,T)) + f(S,T)\psi(1,S) = 0. \tag{22}$$

In general this expression may be hard to solve analytically but a numerical solution for $S^*(T)$ can be implemented using a bracketing zero finder. Given such an $S^*(T)$, we may write the trader's expected utility as a function of $T$ alone and optimize from there to find the equilibrium.

Without solving the model fully, we can make a few general statements about the solution. First, there are reasonably regularity conditions on the distribution $F(\cdot, T)$ which make it so that the transaction will fail with nonzero probability in equilibrium. In particular, if $F$ describes an approximately normal distribution, the probability of of tail events falls off according to exponential decay. Therefore, the benefit of additional slack in $S$ which increases the probability that the order goes through is eventually outweighed by the cost of sandwhich induced slippage when $\psi$'s decay is sub exponential as is the case with the constant product AMM, which has polynomially bounded decay. Note that this assumption, while it may seem like a knife edge case is actually not very strong. Recall that the reason we expect to see randomness in reserves is because of the actions of other traders. Therefore, the noise we expect to see is a simple sum of other trader's random actions. Under sufficient regularity conditions on the joint distribution of these trades, CLT will apply and give us an approximately normal distribution.

Moreover, we can say that the relationship between miner tip $T$ and tolerance parameter $S$ is negative since the monotonicity condition on trading noise guarantees that the trader will be able to tighten his slack parameter when noise is reduced. From this we conclude that the price of noise, in equilibrium is negative. In other words given two states of the world, one with higher noise, and another with lower noise, the trader will choose the lower noise world and in fact the utility difference between the two worlds is separable.

## 5.1 Full Model

The goal in the full model is to introduce many traders who produce the noise we were describing in the previous model. $N$ traders arrive with an urgent inelastic need to trade either one unit of token 0 or one unit of token 1. They simultaneously sign and submit orders to the mempool with a miner tip $T$ and a slippage tolerance $S$. The miner will gather transactions in order of the miner tip and process them in serial miner tip order, possibly inserting sandwiching transactions when they are profitable. If executing the order would mean that the slippage tolerance is too high, the order is canceled and the trader receives a utility of $-T$. He receives $\psi(1, S, R_2 - \psi(S - R_1, R_1, R_2)) - T$ if the order executes but is frontrun, and $\psi(1, R_1, R_2)$ if the order executes and is not frontrun. The sandwicher pays $2T$ to sandwhich an order with miner tip $T$ representing the opportunity cost of including 2 additional blocks with miner tip $T$.

**Theorem 5.1.** The model has no pure strategy equilibria.

*Proof.* Consider a potential pure strategy equilibrium in which each agent submits the same miner tip $T$. Then each agents faces about half of the total uncertainty induced by the process, since each other trader goes before them with probability $1/2$. By submitting a miner tip $T + \epsilon$ a trader can profitably deviate and set slippage tolerance to 0. Because of the separability result discussed in the previous model, there must exist an $\epsilon$ small enough that this deviation is profitable in expectation. So full pooling is not an equilibrium.

Suppose there is a pure strategy equilibrium in which there is some separation in tip size. That is for some $i, j$ we have $T_i > T_j$ and without loss of generality there is no agent $\ell$ such that $T_i > T_\ell > T_j$. Then there is an $\epsilon$ sufficiently small such that setting $T_i' = T_i - \epsilon > T_j$ is a profitable deviation since trader $i$ will observe the same noise level for a lower price. Notice that although miner tip can also be a security parameter that disincentives sandwiching. The miner tip of the agents with the lowest tip $T_N$ must already be the solution to the optimization problem in the first model given the full noise. Since $T_i > T_j \geq T_N$, we know that increasing miner tip above $T_j$ without reducing noise level reduces expected utility for agent $i$. So neither a fully pooled equilibrium nor an equilibrium with any separation can exist. □

Notice that the barrier for the fully separating equilibrium was that miner tips existed on a continuum but there were only finitely many traders. Thus if we want a Nash equilibrium we must do something to fill in the gaps such as introduce a continuum of trader types.

Alternatively, the fully separating equilibrium is a Riley equilibrium [10].

## 6 Frequent Batch Auctions on Chain

As we have seen, frontrunning induces welfare loss as a downstream effect of the provisions that traders take to protect themselves. A remarkably simple solution presents itself which completely eliminates frontrunning. Instead of processing orders in serial within blocks, why not process them all at once?

In particular, we will accept orders of the same type as an AMM would: a tuple of (x,S), where x is an amount and S is the slippage tolerance. As orders are processed, the smart contract will increment an internal sum by $x$, this keeps track of the total directional volume so far after matching. Then the order is inserted into the correct spot in a list sorted by $S$ then $x$. It keeps two separate lists, one for buyers and one for sellers. The first time the contract is called in a subsequent block it resolves these trades in the following manner. It checks if the full order can be resolved on a permissioned AMM that only it can transact with using the leading entry in the list of slippage tolerances as slippage tolerance. If it cannot be resolved, the leading transaction is removed from the list and the internal sum is decremented by the size of the removed transaction. The process repeats until the order resolves or there are no more transaction in either list.

**Theorem 6.1.** Frontrunning is not profitable under this mechanism.

*Proof.* Addition is commutative. □

## 7 Discussion

The vast majority of effort devoted to addressing Miner Extracted Value has been focused on underlying consensus and blockspace markets. I argue that the root of the problem is actually poorly designed exchanges and that most problems which appear to be at the consensus or blockspace market level are actually the downstream effect of these market failures.

The nice thing about the mechanism proposed in this paper is that it does not introduce much complexity beyond what would be required on a central limit order book or modern AMM based equivalent like Uniswap v3. In fact it's probably simpler for users to understand than Uniswap v3. Moreover, it is efficient enough that it, or a nearby algorithm, could actually be implemented on chain. Unlike in traditional markets, there is no need to get approval from all of the stakeholders to change the fundamental mechanisms in play, all that is needed is for the contracts to be written and deployed.

## References

[1] E. Budish, P. Cramton, and J. Shim, " The High-Frequency Trading Arms Race: Frequent Batch Auctions as a Market Design Response *," *The Quarterly Journal of Economics*, vol. 130, no. 4, pp. 1547–1621, Jul. 2015, ISSN: 0033-5533. DOI: `10.1093/qje/qjv027`. eprint: `https://academic.oup.com/qje/article-pdf/130/4/1547/30637414/qjv027.pdf`. [Online]. Available: `https://doi.org/10.1093/qje/qjv027`.

[2] M. Neuder, D. Moroz, R. Rao, and D. Parkes, "Defending against malicious reorgs in tezos proof-of-stake," Sep. 2020.

[3] Q. Wang, R. Li, Q. Wang, S. Chen, and Y. Xiang, *Exploring unfairness on proof of authority: Order manipulation attacks and remedies*, 2022. DOI: `10.48550/ARXIV.2203.03008`. [Online]. Available: `https://arxiv.org/abs/2203.03008`.

[4] L. Zhou, K. Qin, and A. Gervais, "A2MM: mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges," *CoRR*, vol. abs/2106.07371, 2021. arXiv: `2106.07371`. [Online]. Available: `https://arxiv.org/abs/2106.07371`.

[5] A. Judmayer, N. Stifter, P. Schindler, and E. Weippl, *Estimating (miner) extractable value is hard, let's go shopping!* Cryptology ePrint Archive, Report 2021/1231, `https://ia.cr/2021/1231`, 2021.

[6] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, "High-frequency trading on decentralized on-chain exchanges," *CoRR*, vol. abs/2009.14021, 2020. arXiv: `2009.14021`. [Online]. Available: `https://arxiv.org/abs/2009.14021`.

[7] G. Angeris, A. Agrawal, A. Evans, T. Chitra, and S. Boyd, *Constant function market makers: Multi-asset trades via convex optimization*, 2021. DOI: `10.48550/ARXIV.2107.12484`. [Online]. Available: `https://arxiv.org/abs/2107.12484`.

[8] *Welcome to flashbots*. [Online]. Available: `https://docs.flashbots.net/`.

[9] H. Berg and T. A. Proebsting, "Hanson's automated market maker," *The Journal of Prediction Markets*, vol. 3, no. 1, pp. 45–59, 2009.

[10] J. G. Riley, "Informational equilibrium," *Econometrica*, vol. 47, no. 2, pp. 331–359, 1979, ISSN: 00129682, 14680262. [Online]. Available: `http://www.jstor.org/stable/1914187` (visited on 05/10/2022).