Conceptual Flaws of Decentralized Automated Market Making

Andreas Park[*]

April 19, 2022

Abstract

Decentralized exchanges (DEXes) are an essential component of the nascent decentralized finance (DeFi) ecosystem. The most common DEXes are so-called automated market makers (AMMs), smart contracts that pool liquidity and process trades as atomic swaps of tokens. AMMs price transactions with a deterministic liquidity invariance rule that only uses the AMM's token deposits as inputs and that has no precedent in traditional finance. Yet in the context of transparent and open blockchain operations, any liquidity invariance pricing function allows so-called sandwich attacks (akin to front-running) that increase the cost of trading and threaten the long-term viability of the DeFi eco-system. Linear price rules that commonly emerge in economic models have similar problems, except for uniform pricing for which sandwich attack profits are limited and smaller, but which invites excessive order splitting.

[*]University of Toronto `andreas.park@rotman.utoronto.ca`. I am grateful for financial support from the *Global Risk Institute* and the *Social Sciences and Humanities Research Council* and to Cam Harvey, Julien Prat, Agostino Capponi, Christine Parlour, Alfred Lehar, Fas Qureshi, Will Cong (the editor) and two anonymous for insightful discussions and feedback. This paper was previously circulated under the title "The Conceptual Flaws of Constant Product Market Making."

Public blockchains such as Ethereum are software protocols that allow multiple parties to operate under shared assumptions and data without an institutional reason to trust each other. A blockchain's core function is storing, processing, and transferring digital value, and the ability to exchange tokens is a central function of these networks. In principle, it would be possible to organize crypto-asset trading directly on a blockchain by registering limit orders as "smart contracts." However, this is not practical because each new limit order costs a validation fee. Unmatched orders would also waste resources as they still need to be processed by all 10,000+ network nodes. Lastly, a peer-to-peer network by itself is not a marketplace.

Instead, until recently users could trade blockchain-based items or tokens only on centralized, "off-chain" exchanges, reducing the blockchain to a settlement infrastructure. This changed by mid-2020 with the release of Automated Market Maker (AMM) systems such as UniSwap or SushiSwap, which have seen tremendous user uptake and process billions of dollars worth of transactions every day, often more than the largest centralized exchanges such as Binance and Coinbase.

AMMs use the blockchain's inherent ability to process code and they are a genuinely novel market structure institution. First, AMMs pool liquidity so that liquidity providers do not compete, a stark contrast to stock exchanges where firms make billion-dollar investments to gain nano-second speed advantages. Even unsophisticated token owners can act as passive liquidity providers by contributing their crypto-assets to a liquidity pool, allowing them to earn a market-making income without specialized skills, constant monitoring of markets, or expensive equipment. Second, AMMs do not directly rely on a market mechanism in which the price equilibrates demand and supply. Instead, AMMs use a hard-coded pricing rule. Third, the pricing function that almost all AMMs employ has no precedent in traditional financial markets. Therefore, this novel setup raises questions about the functioning of markets, the informational efficiency of prices, and the possible emergence of arbitrage.

The workings and economic impact of AMMs must be examined against the backdrop of the functioning of a blockchain. As I show in this paper, the organization of block production and the need for the efficient use of resources present significant conceptual challenges for any automated market-making setup. The hard-coded pricing function that swap exchanges currently use creates persistent "exploit" opportunities, imposes excess costs on users, congests the network, and raises transactions costs for all blockchain users. The challenge is fundamental: no known pricing function can satisfy a set of reasonable requirements.

A swap exchange/AMM creates a liquidity pool by combining deposits of pairs of tokens A and B from liquidity providers. A liquidity demander can trade against this pool by sending one type of token and thus receiving the other token in an atomic swap. A pricing rule determines the exchange rate. Most AMMs that are in operation are designed to be self-contained in the sense that the price keeps the pool's liquidity invariant: when a liquidity demander removes one type of token from the pool, they have to deposit a quantity of the other type of token such that the aggregate liquidity of the pool defined by a "bonding curve" remains unchanged.

The specific bonding curve that AMMs such as UniSwap or SushiSwap use is referred to as a "constant product" pricing rule.[1] Suppose the pool contains X units of

[1]The first mention of automated decentralized market making is in a 2016 Reddit post by Vitalik Buterin. Martin Koppelmann proposed the first constant product pricing scheme.

token A and Y units of B. Under the constant product rule, the pool's liquidity c is the product $X \cdot Y = c$. To remove Q of the A tokens, a buyer must add P of the B tokens such that the liquidity level remains invariant, $c = (X - Q) \cdot (Y + P)$.

Let me explain the problem with this functional form using specific numbers. At the beginning of February 2022, the UniSwap (V2) token pair ETH (the native cryptocurrency of Ethereum) and USDC (a digital representation of the US dollar) contained approximately 38,100 ETH and 118M USDC; the implied marginal price of 1 ETH was thus around \$3,097. A liquidity demander who wants to buy 100 ETH from this contract would pay approximately \$3,105 per ETH. However, suppose a speculator manages to inject a same-sized trade before this 100 ETH trade and then reverses it immediately after. What would this speculator earn from "sandwiching" the original trade? The initial trade would cost the speculator \$3,105 per ETH, and reversing the trade yields \$3,121 per ETH, for a total profit of \$1,639 — which is value extracted from the original trader. Although this is a lot of money, the loss is only 5 basis points of the transaction size, which may be bearable. But it gets worse: had the attacker submitted a "sandwiching" trade of 1,000 ETH, the value extracted would be almost \$17K, and had they submitted a trade for 10,000 ETH, it would have been \$261K. This problem is pervasive: as I show in this paper, for a general class of pricing functions that are based on liquidity invariance (such as the constant product rule), under a mild convexity assumption, sandwich attacks are profitable for any trade (modulo fees), and the attacker's profits are unbounded.

Although sandwich attacks are profitable, are they possible? The answer is affirmative because of the inherent transparency of blockchain transaction processing.

Namely, signed and processed blockchain transactions wait in publicly visible "mem-pools" for validators to include them in a block. A front-runner can always add a new pair of transactions to the mem-pool to sandwich a trade. Crucially, the trade price is not fixed when it is submitted, nor when it is added to the mempool. What determines the price of a trade is the pool's liquidity at the time when a validator processes it within a block. A trade's cost thus depends even on its position in the block, which allows a sandwich attack to be successful and profitable.

The sandwich attack problem is not hypothetical, and I am not the first to highlight it.[2] The blockchain community classifies sandwich attacks as a form of "(miner) extractable value" (MEV), i.e., rents that a third party can extract from the original initiator. According to the Flashbot tracker, automated market makers account for almost 80% of the roughly $600M of observed MEV.[3]

The blockchain community has developed tools and patches to mitigate sandwich attacks, and AMMs themselves have features that limit the losses that a trader would incur; I review several approaches in the paper. They do not, however, resolve the problem and attacks continue to occur systematically. As anecdotal evidence, there is an Ethereum address with the DNS designation "MEV Bot: 0x000...94e" that engages in sandwich attacks many times each day. For instance, in block 14404142 (on March 17, 2022), this bot sandwiched a UniSwap transaction of ETH-APE (the governance

[2]See, e.g., Vitalik Buterin's post here. Angeris and Chitra (2021) formalize it as "path deficiency."

[3]Sandwich attacks may not cause the entire amount. It's important to recognize that MEV creates externalities beyond the value that users lose because front-running bots regularly bid up network usage fees in so-called priority gas auctions, which raise the price of Ethereum transactions for all other users. Running data is available at ETH Gas Station; UniSwap's V2's main contract address is 0x7a250d5630b4cf539739df2c5dacb4c659f2488d. In a recent paper, Gans and Holden (2022) propose a mechanism design approach to prevent third-party value extraction. The solution, however, does not apply to and resolve the sandwich attack problem. Instead, it applies to the type of cases that the now-famous blog post Ethereum is a Dark Forest refers to.

token of the "Bored Ape Yacht Club").[4] The original trader wanted to swap 1.5 ETH for APE tokens but got sandwiched. The attacker spent 4.14 ETH to buy 1,668 APE and reversed this trade for 4.46 ETH right after for a profit of 0.3 ETH. Notably, this sandwich attack was submitted using the Flashbot protocol, one of the countermeasures that are in place to *prevent* such a sandwich attack.

The constant product pricing rule is ad hoc, not an equilibrium outcome, and without precedent in financial markets. That raises the question: how would the pricing rules from economic models perform in a blockchain environment? To answer this question, I focus on linear price functions as are common in the economic literature, e.g., Kyle (1985), Grossman and Miller (1988), or Biais (1993). Linearity arises often, either by design or because non-linear prices cause distortions. My analysis shows that such rules don't perform much better. The linear rule that resembles the pricing of a limit order book allows unbounded sandwich attack profits, too. A uniform pricing rule does somewhat better because it intrinsically limits the attacker's profit. However, this pricing function incentivizes excessive order splitting, which would congest the network.

In the absence of a clear theoretical ranking, the question remains if one pricing rule would be better for traders in practice, e.g., because it is cheaper. To enable a fair comparison, I develop a model of liquidity provision that can accommodate different pricing functions. I then calibrate the model's parameters using data from observed AMM trading. For liquid trading pairs, constant product prices are a little lower, whereas traditional pricing for illiquid pairs is notably cheaper. Losses from sandwich trades are lowest for the uniform pricing rule.

[4] The relevant three transactions are under the following links: URL link 1, 2, and 3.

Related Literature. There are three other theoretical papers on decentralized exchanges. These three model trading under asymmetric information. In the traditional microstructure literature, the price endogenously reflects the information, whereas with constant product pricing the functional form is fixed. Market makers determine the pricing function's slope with their ex ante supply of liquidity, and in their decision must account for the rewards they receive given the information shocks.

Lehar and Parlour (2021) use two assumptions to determine the AMM liquidity provision under asymmetric information. First, they assume zero profits for liquidity providers. Second, informed traders submit fixed size quantities (because of fixed-size shocks to the fundamental). They show that for many parametric configurations, investors do not prefer the limit order market over constant product pricing.

Aoyagi and Ito (2021) study prices when centralized and decentralized swap exchanges co-exist in a model with asymmetric information. They show that the AMM price often deviates from the fundamental value even when information can be perfectly inferred from the price.

Capponi and Jia (2021) examine the impact of price volatility, caused by trading on centralized exchanges, on the welfare of liquidity providers in automated swap exchanges for a broad set of functions (twice-continuously differentiable, and convex). They identify the conditions for which there is a breakdown of liquidity supply in the automated system and they show that a pricing curve with larger convexity reduces arbitrage rents, but also decreases trading activities.

The main interest of this paper are the properties of the functional forms of different pricing rules. The model in Section III. of this paper additionally studies the liquidity provision decision of risk-averse market makers under different pricing functions. It is based on symmetric information, thus complementing the literature.

I. Properties of Pricing Functions

A. *Liquidity Invariance Functions*

The idea of an automated pricing rule for a liquidity pool is that as a liquidity demander removes one type of token from the pool, they have to deposit a quantity of the other type of token such that the liquidity of the pool, as defined by the so-called "bonding curve," remains unchanged. The ubiquitous constant product rule is one example: A smart contract contains deposits of X units of A tokens and Y units of B tokens. To withdraw (or purchase) quantity Q of A-type tokens, a user deposits a quantity P of B-type tokens. After the trade, the contract contains $X - Q$ of the A tokens and $Y + P$ of the B tokens. The constant product pricing rule requires *liquidity invariance* in the sense that the level of liquidity is defined by a constant c with $c = X \cdot Y$, so that when trading quantity Q, P satisfies $c = (X - Q)(Y + P)$.

A more general way to think about pricing in this context is that the exchange rate of tokens follows a *liquidity invariance condition* in the sense that there is a function $L(X, Y)$ such that for demanded quantity Q, when the pool contains X of the A and Y of the B tokens, quantity $P(Q|X, Y)$ satisfies

$$L(X, Y) = L(X - Q, Y + P(Q|X, Y)).$$

In what follows I will use superscript li to signify liquidity invariance pricing generally and cp for constant product pricing specifically. Moreover, prices and values in economic models are measured in a numeraire, which I assume here to be the B token. I will further assume that the A token has a fundamental value V and that absent any trades, the marginal price of an A token coincides with its fundamental value. In what follows, therefore, for known V, Y is fixed by X. To reduce the number of displayed variables and thus to simplify the exposition, I will omit Y when referring to a pricing function and only display changes to the number of A tokens. I will also omit reference to the token inventories altogether for the status quo; therefore $P^{\mathsf{li}}(Q|X,Y) = P^{\mathsf{li}}(Q|X) \equiv P^{\mathsf{li}}(Q)$. The constant product rule resembles a Cobb-Douglas production function in the sense that the input token amounts X and Y "produce" liquidity according to $L(X,Y) = X^{\alpha} \cdot Y^{\beta}$.

In this remainder of this section and the next, I will explore the properties of pricing functions that emerge from liquidity invariance conditions and compare them to linear pricing functions. I begin by imposing assumptions that are common for production/consumption functions in consumer and producer theory. Namely, I assume strict *convexity to the origin* in the sense of diminishing payments received: as users deposit A token in the contract, the amount of B tokens received in return diminishes, and vice versa. In a blockchain environment, it is further important that the price for Q units of the A token in terms of P units of the B tokens is determined by a function which is *continuous* and *differentiable.*

B. Linear Pricing Functions

The constant product rule is an exogenous rule. In contrast, in economic models, in particular in finance, a price usually arises endogenously as it equilibrates the choices of optimizing consumers and producers. Moreover, in many market microstructure trading models, the equilibrium pricing rule is a linear function of demand; examples are Kyle (1985) who solves his model assuming a linear price function, and rational expectations models in the tradition of Grossman and Miller (1988) where a linear price function emerges in equilibrium. Linear prices also arise in many other economic models in the literature; e.g., linear schedules are critical in optimal taxation as they prevent disincentives. Linear marginal price functions are, therefore, an important class in economics.

Applied to a liquidity pool, one implicitly assumes —as is common in most economic models— perfect capital markets in the sense that liquidity providers are able to borrow and then supply any funds at market rates. In other words, the deposited quantities X and Y of the tokens are not part of the pricing rule so that these models do not produce an invariance rule.

Henceforth, I use the term “pricing function” $P(Q|I)$ for the cost of a trade Q of the A token when liquidity providers' aggregate inventories in the A token are I. I further use $p(q|I)$ for the price of the q-th unit and refer to p as the marginal price function. Since the asset (or A token) has expected value V, I require $p(0|0) = V$. A linear price would therefore take the form

$$p(q|I) = V + mq + nI, \quad m > 0 > n, \tag{1}$$

where the restriction on the parameters m, n stems from the fact that prices increase in quantities and decrease in inventories (assuming liquidity providers are risk-averse). There are two common ways to compute the total cost of an order $P(Q|I)$:

$$P^{\mathsf{up}}(Q|I) = Q \cdot p(Q|I) = Q(V + m^{\mathsf{up}}Q + n^{\mathsf{up}}I) \quad \text{and} \tag{2}$$
$$P^{\mathsf{dp}}(Q|I) = \int_0^Q p(q|I)dq = Q\left(V + \frac{m^{\mathsf{dp}}}{2}Q + n^{\mathsf{dp}}I\right). \tag{3}$$

The first is the most common price-times-quantity rule where the cost of the entire order is determined by the price of the marginal unit Q. This rule is sometimes referred to as *uniform pricing*, hence the superscript up.

The second rule (3) arises in limit order markets where the price of a large order is the sum of smaller trades that execute against standing limit orders which each can have a different price. This rule is known as *discriminatory pricing*, hence the superscript dp.

Lemma 1: *The linear prices are continuous iff* $n^{\mathsf{up}} = -m^{\mathsf{up}}$ *and* $n^{\mathsf{dp}} = -2m^{\mathsf{dp}}$.

Proof. Continuity after a trade of q implies that the marginal pricing functions adjust by the amount of the new inventory, $p(q|I) = p(0|I-q)$. This means in particular that $V + mq + nI = V + n(I-q)$. For (2) and (3) this implies respectively, $n^{\mathsf{up}} = -m^{\mathsf{up}}$ and $n^{\mathsf{dp}} = -2m^{\mathsf{dp}}$. □

In what follows I will use superscript lp to signify a linear pricing rule generally. As with invariance pricing, I will also omit reference to the inventory altogether for the status quo; i.e., $P(Q|I) \equiv P^{\mathsf{lp}}(Q)$.

C. Desirable Properties and Institutional Features of Blockchain Operations

Efficient Use of Computing Resources. Blockchains such as Ethereum are computer networks that guarantee the execution of code and can thus run an automated market maker (decentralized) application. However, blockchains as distributed networks require that all validator nodes execute every piece of code to ensure that the state of the blockchain is updated correctly. Moreover, users have to pay the validator for the computational cycles ("gas") that their code consumes. Thus, avoiding economically valueless code executions is individually rational and socially desirable.

In the context of trading, it is therefore desirable if a pricing function has *size consistency* so that splitting an order into two (or more) orders costs no less than sending a single order of the same size, ignoring blockchain processing costs.

Definition (Size Consistency): *Price functions* $P^{\mathsf{li}}, P^{\mathsf{lp}}$ *satisfy* size consistency *when order splitting is not profitable, i.e., if for any* $\alpha \in (0,1)$, *trade size* $Q < X$ *holds* $P^{\mathsf{lp}}(\alpha Q) + P^{\mathsf{lp}}((1-\alpha)Q | I - \alpha Q) \geq P^{\mathsf{lp}}(Q)$ *and* $P^{\mathsf{li}}(\alpha Q) + P^{\mathsf{li}}((1-\alpha)Q | X - \alpha Q) \geq P^{\mathsf{li}}(Q)$ *respectively.*

Proposition 1 (Size Consistency of Traditional and Liquidity Invariant Pricing):

1. *Liquidity invariance based pricing rules always satisfy size consistency.*
2. *Among linear functions, discriminatory pricing satisfies size consistency, uniform pricing does not.*

Proof. of 1. The statement is a direct consequence of pricing along an iso-liquidity curve and best explained graphically. Figure 2 shows that as a user removes quantity Q, they must pay P such that the point $(X - Q, Y + P)$ is on the iso-liquidity curve L.

Splitting up Q into parts $\alpha Q, (1-\alpha)Q$ simply means splitting the amount P that the liquidity demander pays.

Proof of 2. Order splitting is unprofitable if and only if $P^{\mathsf{lp}}(\alpha Q) + P^{\mathsf{lp}}((1-\alpha)Q|I - \alpha Q) \geq P^{\mathsf{lp}}(Q)$ for all $\alpha \in (0,1)$. Substituting in the two functional forms (2) and (3) of P^{lp}, this relationship holds if $n \leq -2m$. In the uniform case, $n^{\mathsf{up}} = -m^{\mathsf{up}}$ and therefore size consistency does not apply. For the discriminatory case, $n^{\mathsf{dp}} = -2m^{\mathsf{dp}}$ and therefore size consistency is satisfied. □

Note further that invariance pricing also implies that buying Q and then selling it right away yields a profit of 0, $-P^{\mathsf{li}}(Q) - P^{\mathsf{li}}(-Q|X-Q) = 0$.

No Sandwich Attacks. In public blockchains, verified but non-settled transactions are added to public "mem-pools" from which validators select transactions for inclusion in a new block. Transactions in mem-pools are publicly visible, and there are bots that monitor mem-pools for non-settled transactions that they can profit from. The type of exploit transactions that are relevant for this paper are so-called "sandwich attacks." A sandwich attack occurs when someone (usually a bot) submits an order which trades before the original one and which the person/bot reverses right after the original trade. The original trade is therefore "sandwiched" between the bot's buy and sell trades.

For the attack to be successful, the attacker needs to be able to control the timing of the two trades relative to the sandwiched one. In the blockchain world, the attacker can control the timing through the gas fees that they pay the validators because most protocols mechanically prioritize transactions by the (gas) fees.[5] Figure 1 illustrates

[5]For instance, in the Parity mining protocol transactions are ordered by their fee (the link points directly to the portion of the code that describes this feature). Of course, miners can modify their code in any way they choose.

the mechanics of these sandwich attacks. Here I study sandwich attack profits modulo fees; Section IV.C. studies the importance of gas fees in detail.

Although a sandwich attack is always *possible*, the question is whether there is always a *profitable* attack. For if there is not, then, arguably, sandwich attacks should not occur. The critical question is therefore whether or not the pricing rule makes sandwich attack intrinsically profitable. In line with the practice in the blockchain community, I refer to sandwich attack profits as *Miner Extractable Value (MEV)*, though I note that any third party and not just miners can extract value with a sandwich attack.

For trade $Q' \in (0, X)$ and pricing functions $P^{\mathsf{lp}}, P^{\mathsf{li}}$, the Miner Extractable Value (MEV) of a sandwich attack of size Q with $Q' + Q < X$ is defined as

$$\begin{aligned} \text{MEV}^{\mathsf{lp}}(Q, Q') &= -P^{\mathsf{lp}}(Q) - P^{\mathsf{lp}}(-Q|I - Q - Q'), \qquad (4) \\ \text{MEV}^{\mathsf{li}}(Q, Q') &= -P^{\mathsf{li}}(Q) - P^{\mathsf{li}}(-Q|X - Q - Q'). \qquad (5) \end{aligned}$$

The first terms $P^{\mathsf{lp}}(Q)$ and $P^{\mathsf{li}}(Q)$ are what the attacker pays on the first leg of the sandwich trade. The second terms, $-P^{\mathsf{lp}}(-Q|I - Q - Q')$ and $-P^{\mathsf{li}}(-Q|X - Q - Q')$ are what the attacker receives after reversing their original trade of Q which they do after the sandwiched trade of size Q' has been processed.

Definition (Absence of Profitable Sandwich Attacks): *A pricing rule P is immune to* sandwich attacks *if for any trade $Q' \in (0, X)$ and all Q with $Q' + Q < X$ holds $MEV(Q, Q') \leq 0$.*

II. Impossibility Results

This section highlights the difficulties in AMM design that arise in a blockchain environment. The first problem relates to pricing functions that are derived from liquidity invariance rules. They always allow intrinsically profitable sandwich attacks, and these profits are not bounded.

Theorem 1: *Any price function P derived from a liquidity invariance condition $L(X,Y) = L(X-Q, Y+P)$ with the properties specified in I.A. allows profitable sandwich attacks and unbounded profits for the attacker.*

Proof. The attacker pays $P^{\mathsf{li}}(Q)$ and receives $-P^{\mathsf{li}}(-Q|X-Q-Q')$. The definition of liquidity invariance implies that the price one pays for buying Q and the price received for selling it right back after are the same. Therefore

$$-P^{\mathsf{li}}(-Q|X-Q-Q') = P^{\mathsf{li}}(Q|X-Q'). \tag{6}$$

Therefore the MEV for the sandwich attack from (5) is

$$\text{MEV}(Q,Q') = P^{\mathsf{li}}(Q|X-Q') - P^{\mathsf{li}}(Q) > 0,$$

where the last inequality follows from the fact that the price is increasing when the inventory in A tokens drops and in B tokens rises.

The increasing, unbounded nature of attack payoffs follows using Figure 3: The attack is of size Q. Its cost is $P^{\mathsf{li}}(Q)$, its revenue $-P^{\mathsf{li}}(-Q|X-Q'-Q)$, the net amount is the MEV. In the figure, the MEV is the difference of the red line and the grey, $P^{\mathsf{li}}(Q'|X-Q) - P^{\mathsf{li}}(Q')$. The latter component is constant. The first component is increasing in Q because the bonding curve $L(\cdot)$ is convex to the origin. $\square$

The next question that arises is whether traditional linear pricing rules fare better than liquidity invariance rules. The answer here is negative, too, as they also allow profitable sandwich attacks. However, there is one positive insight: there are parametric configurations such that the profits from front-running are bounded. These come with the caveat that they do not satisfy size consistency.

Theorem 2: *Let the marginal price be $p(q|I) = V + mq + nI$ with $m > 0 > n$.*

1. *Any such linear pricing rule allows profitable sandwich trades.*
2. *Linear pricing rules that satisfy size consistency allow unlimited front-running profits. Linear rules that allow only limited front-running profits do not satisfy size consistency.*

Proof. for 1.: Let the sandwiched trade be of size $Q' > 0$. Note that the uniform and discriminatory pricing rule both in (2) and (3) both have the same functional form $p(q|I) = V + mq + nI$ subject to the transformation of $m = m^{\mathsf{up}}$ and $m = m^{\mathsf{dp}}/2$. Then front-running by amount Q pays

$$\begin{aligned}
\mathrm{MEV}^{\mathsf{lp}}(Q, Q') &:= -P^{\mathsf{lp}}(Q) - P^{\mathsf{lp}}(-Q|I - Q' - Q) \\
&= -Q \cdot (m \cdot Q + n \cdot I) + Q \cdot (-m \cdot Q + n \cdot (I - Q' - Q)) \\
&= -Q^2(2m + n) - n \cdot Q \cdot Q'. \qquad (7)
\end{aligned}$$

This expression is a quadratic equation in Q and has a local minimum when $2m+n < 0$, and it has a local maximum when $2m+n>0$. Furthermore it is a linearly increasing function for $2m+n=0$. The arg-min or arg-max is $Q^* = -\frac{1}{2}\frac{nQ'}{2m+n}$, and the value of (7) at that point is $1/4n^2{Q'}^2/(2m+n)$.

Case 1: Q^* is a minimum. In this case $Q^* < 0$. As MEV^{lp} is a quadratic, increasing function for all $Q>0$, for large enough Q, MEV^{lp} is positive. Moreover, MEV^{lp} is an unbounded function.

Case 2: Q^* is a maximum. Then $Q^* > 0$. Furthermore, the MEV^{lp} function $-Q^2(2m+n) - nQQ'$ is positive for all $Q \in \left(0, \frac{-nQ'}{2m+n}\right)$.

Proof. for 2.: By Lemma 1.2, order splitting is not profitable for discriminatory pricing when $n=-2m$, but it is profitable for uniform pricing, $n=-m$. For $n>-2m$, the quadratic function for MEV^{lp}, $-Q^2(2m+n)-nQQ'$, has an internal maximum for a positive value of Q for every Q'. For $n=-2m$, $-Q^2(2m+n)-nQQ'$ MEV^{lp} is a linear, unbounded function. □

Note that for $n \nearrow 0$, MEV^{lp} vanishes. However, this would also imply that continuity is violated; economically, $n \nearrow 0$ implies that inventories play no role and that trades have no price impact.

III. Empirical Comparison of the Pricing Models

A. *Overview.*

The conclusion for the analysis so far is that the common pricing functions that I discuss here have deficiencies as they all allow profitable sandwich attacks, though at least under uniform pricing, these profits are bounded. This raises the question if any of the pricing functions is at least empirically better, e.g., because it is cheaper.

My goal is to determine the prices that the same liquidity providers would produce when faced with different pricing rules. To tackle this question I proceed as follows: I first develop a single model of liquidity provision that allows an analysis using the different pricing functions. I then calibrate this model to a small dataset from UniSwap.

The model allows me to determine how the liquidity providers that we can implicitly observe from constant product pricing data would act if the pricing rule was traditional, so that I can then perform a fair comparison of the three approaches.

B. The Liquidity Provision Decision

Market participants who engage in automated market making are producers of liquidity. Their costs are determined by the risk that they absorb from liquidity demanders, and their income is determined by the price that they can charge.

Market participants trade token A which has an intrinsic value with mean V and variance σ^2 against a token B which, for simplicity, is assumed to be a numeraire such as a fiat currency or a stablecoin. All information is public. Tokens are infinitely divisible. Liquidity demand Q is exogenous, inelastic, and uniformly distributed, $Q \sim U[-L, L]$.

There are N identical, price-taking liquidity providers. In a world with automated market making, liquidity providers make an ex-ante commitment to provide a fraction α_i of the liquidity. Thus, when liquidity demand is $Q \sim U[-L, L]$, an intermediary absorbs fraction $\alpha_i Q$ and receives a payment $\alpha_i P$ in return. To capture the costs associated with absorbing a risk, liquidity demanders maximize a mean-variance utility of terminal wealth as follows

$$\max_{\alpha} \int_{-L}^{L} \left(V(I_i - \alpha q) - \frac{\kappa \sigma^2}{2} (-\alpha q + I_i)^2 + \alpha P \right) \frac{1}{2L} \, dq. \tag{8}$$

This model is loosely based on Grossman and Miller (1988) and Biais (1993).[6]

C. Optimal Liquidity Provision in a Traditional Model

In the traditional model, the equilibrium pricing function ensures that markets clear, i.e., the price is such that for any Q, liquidity providers are willing to supply fraction $\alpha_i Q$ and the liquidity supplied coincides with the liquidity demanded, $\sum_{i=1}^{N} \alpha_i Q = Q$. Section I.B introduces linear pricing rules. The following result describes the representations of m and n for the linear pricing rules that result from liquidity providers' optimization in the model specified above.

Lemma 2 (Equilibrium Price in Traditional Market Making): *Define $\ell = \frac{\kappa\sigma^2}{N}$ and assume pricing functions are continuous. To buy Q units of token A from the N intermediaries, the equilibrium coefficients for $P^{\mathsf{up}}(Q)$ and $P^{\mathsf{dp}}(Q)$ are $m^{\mathsf{up}} = -n^{\mathsf{up}} = \ell$ and $m^{\mathsf{dp}} = -n^{\mathsf{dp}} = 2\ell$.*

Proof. Part 1: Determining m. The result follows by verifying that the posited functions are indeed solutions to (8). Starting with the uniform price approach, and sub-

[6]In, Biais (1993) the fundamental value is normally distributed and liquidity providers have a CARA utility of terminal wealth which together gives rise to a mean-variance expected utility function. Liquidity suppliers in Biais (1993) choose the quantity that they will provide after observing the liquidity demand. Here, they need to commit ex ante, and there is uncertainty about the respective quantities. The uniform-pricing equilibrium price that I derive below coincides with the one in Biais (1993), but the CARA-normal formulation is not solvable in closed form for the constant product price. To preserve comparability, I choose the simpler, direct mean-variance formulation.

stituting into (8), the first order condition is

$$\frac{L^2}{3}(\alpha\kappa\sigma^2 - m) = 0.$$

Therefore, the optimal α is

$$\alpha = \frac{m}{\kappa\sigma^2}.$$

Since liquidity providers are identical, in equilibrium it must hold that $\alpha = 1/N$. Using this fact, m is defined as

$$m^{\mathsf{up}} = \frac{\kappa\sigma^2}{N}.$$

Next, for the discriminatory rule, the first order condition for (8) is

$$\frac{L^2}{3}(\alpha\kappa\sigma^2 - m/2) = 0.$$

Therefore, the optimal α is

$$\alpha = \frac{1}{2}\frac{m}{\kappa\sigma^2}.$$

Since it must hold that $\alpha = 1/N$,

$$m^{\mathsf{dp}} = \frac{2\kappa\sigma^2}{N}.$$

Part 2: Determining n. Lemma 1 shows that for the two pricing functions to be continuous, it must hold that $n^{\mathsf{up}} = -m^{\mathsf{up}}$ and $n^{\mathsf{dp}} = -2m^{\mathsf{dp}}$. Therefore, $n^{\mathsf{up}} = -m^{\mathsf{up}} = -\ell$ and $n^{\mathsf{dp}} = -2m^{\mathsf{dp}} = -2\ell$. □

Notably, for a starting inventory of $I = 0$, the ex ante prices and therefore the ex ante revenue for liquidity providers are identical for the two linear pricing approaches.

D. *Optimal Liquidity under Constant Product Pricing*

Section I. introduces constant product pricing as a special case of liquidity invariance pricing: Under constant product pricing, a liquidity demander who wants to purchase $Q \in (-\infty, X]$ of the A token, has to deposit P units of the B tokens into the AMM contract such that P solves $(X - Q) \cdot (Y + P) = X \cdot Y$. Therefore

$$P^{\mathsf{cp}}(Q|X) = P^{\mathsf{cp}}(Q) = Q \cdot \frac{Y}{X - Q} = Q \cdot \frac{VX}{X - Q}, \tag{9}$$

where the last equality stems from the fact that the marginal price must coincide with the fundamental value, $V = Y/X$.

Under traditional pricing, the market clearing condition would determine the quantity that liquidity providers deliver, and liquidity providers implicitly commit to provide arbitrarily many tokens. The price of Q is determined by slope parameters that reflect the liquidity providers' risk aversion. Under constant product pricing, the quantity of supplied and demanded tokens cannot coincide because the price is not defined for $Q \to X$ ($\lim_{Q \to X} P^{\mathsf{cp}}(Q) = \infty$).

Instead, the approach is more subtle. For fixed V (and thus Y is determined by V and X), the slope of the pricing curve is driven by the ex ante quantity of A tokens, X, and therefore the aggregate quantity supplied determines the price that liquidity providers receive in equilibrium. In equilibrium, this amount has to be the size such that each liquidity provider is willing to contribute a share so that the aggregation of these shares is indeed the size that gave rise to the price function.

Formally, in the optimization problem, price taking liquidity providers take the functional form and thus X as given and decide which fraction α_i of X they are willing to supply. The equilibrium quantity of liquidity X is then the number of tokens such that liquidity providers are each willing to deposit an amount of liquidity $\alpha_i X$ such that $\sum_{i=1}^{N} \alpha_i X = X$. Since liquidity providers are identical, $\alpha_i = \alpha$. For the constant pricing approach, the optimal α for (8), keeping X fixed is

$$\alpha(X) = \frac{3}{2} \frac{XV}{\ell L^3} \left(X \ln \left(\frac{X+L}{X-L} \right) - 2L \right). \tag{10}$$

Assuming the presence of N identical liquidity providers, the equilibrium quantity X^* must then solve $\alpha(X^*) = 1/N$.

Lemma 3 (Constant Product Liquidity Provision): *For each number of liquidity providers N, there exist an amount $X(N) = \sum \alpha_i X(N)$.*

Proof. For identical liquidity suppliers, $\alpha(X) = 1/N$. The right hand side of (10) goes to infinity for $X \to L$ and to 0 for $X \to \infty$, and thus a value X^* exists such that $\alpha(X) = 1/N$ for any N. □

Therefore, although the constant product pricing rule does not itself emerge from an economic model, one can formulate the liquidity providers' optimization problem and derive an equilibrium quantity of liquidity that is compatible with optimizing behavior. Moreover, by deriving behavior from the same underlying optimization problem, I can now compare the prices for different pricing rules.

E. Empirical Calibration

The goal of this subsection is to provide a meaningful comparison of the prices and MEVs that arise under the difference pricing rules. Instead of using arbitrary values for the various parameters, I believe it may be more insightful to use values that reflect trading as it occurs in practice. Therefore, for this exercise, I use data from UniSwap V2 for two prominent contracts: ETH-USD and BTC-USD.[7] These two pairs are stylized examples because at the time of writing wETH-USDC is the most liquid trading pair in UniSwap V2 whereas wBTC-USDC is a low liquidity pair.

Parameter Derivation. For the comparison, I need to obtain estimates for the expected fundamental value V and the variance σ^2, risk aversion parameter κ, the number of liquidity providers N, the deposited quantity X, and the uniform distribution bounds $[-L, L]$. The derived and estimated parameters are in Table I.

According to UniSwap Analytics, on February 8, 2022, the wETH-USDC pool contained approximately 38,100 ETH and an equivalent amount of USDC for an implied price of \$3,100. The wBTC-USDC contract contained 6.25 BTC and an equivalent amount of USDC for an implied price of \$44,000. These deposited amounts serve as my proxies for X, the quantities of the A token, and I use the implied prices as my proxy for the expected value V.

The traditional market maker model requires a choice for the risk aversion coefficient κ. Using a meta-analysis, Babcock, Choi, and Eli Feinerman (1993) (Table 1)

[7] To be precise, neither ETH nor BTC can be directly traded in AMM systems. Instead, these systems use so-called wrapped tokens. Furthermore, these system don't use the USD, but digital representations in the form of the USDC stablecoin.

report that the CARA coefficient κ falls anywhere between .00001 and 0.5, where most of the values in the literature are at the lower end of this range; in my subsequent calibration, I use $\kappa = 0.0005$.

To estimate the standard deviation on V, I use daily data from CoinDesk (for BTC and ETH) and compute the average standard deviation of daily prices for each month of 2021. For the w-ETH-USDC contract, it is approximately 260, for BTC it is approximately 4100. These values serve as my estimates for σ^2.

The next set of parameters relate to the distribution of quantities q. I obtain the swap transactions for wETH-USDC and wBTC-USDC from Etherscan.io for January 2022. The empirical distribution of quantities resembles a uni-modal function, but since the purpose of this section is to provide an illustration and not a full empirical analysis, I will follow the description from the theoretical model and approach the empirical distribution as uniform. Since the empirical distribution is not uniform, my goal is to choose a bound L that captures many trades so that the uniform assumption is at least loosely justified and so that I do not greatly overestimate the prevalence of large quantities. For wETH-USDC, 92% of observations are smaller than 10 and 52% smaller than 1. For wBTC-USDC, 97% are smaller that 0.5 and 58% smaller than 0.1. These four values are henceforth my estimates for the uniform distribution bound L.

Finally, to obtain the number of liquidity providers, I substitute the estimated parameters into (10) and solve for $\alpha^* = 1/N$. At the observed values for X of 38,100 for wETH-USDC, I estimate N to be 415 for $L = 10$ and 409 for $L = 1$. For wBTC-USDC, I estimate N to be 1.2 for both $L = 0.5$ and $L = 0.1$.

Measures for Comparison. The first key metric in the comparison is the average price; it captures the rents of liquidity providers. Formally, the average price is

$$E[P] = \int_{-L}^{L} P(q)\ f(q)dq. \tag{11}$$

Note that ex ante and for inventory $I = 0$, prices under the uniform and the discriminatory rule coincide.

As shown above, sandwich attack profits are theoretically unbounded for constant product and discriminatory pricing. For uniform pricing, for any Q', the theoretically optimal sandwich attack is a trade of size $Q = Q'/2$. To ensure comparability I will compute sandwiching profits for sandwich attacks with two sizes: the first is for $Q = Q'/2$. The second is for size $Q' = Q$. In the latter case, sandwich trades under uniform pricing are zero. Following Section II., the sandwich attack profits are

$$\text{MEV}^{\mathsf{dp}}(Q, Q') = 2\ell QQ',\ \text{MEV}^{\mathsf{up}}(Q, Q') = \ell Q(Q' - Q),\ \text{and} \tag{12}$$

$$\text{MEV}^{\mathsf{cp}}(Q, Q') = \frac{YQQ'\left(2X - Q - Q'\right)}{\left(X - Q\right)\left(X - Q - Q'\right)\left(X - Q'\right)}. \tag{13}$$

I then compute

$$E[MEV] = \int_{-L}^{L} \text{MEV}\left(\frac{q}{2}, q\right)\ f(q)dq \text{ and } E[MEV] = \int_{-L}^{L} \text{MEV}\left(q, q\right)\ f(q)dq. \tag{14}$$

Results. Table II contains the estimated costs with its rows as in Table I.

For the liquid wETH-USDC pair, the average prices under traditional pricing and constant product pricing differ only in the fourth digit for both the tight and the wide range of quantities, where the constant product price is marginally lower. For the less liquid wBTC-ETH trading pair, traditional pricing is cheaper, by about 20 basis points for the wider range of trade sizes.

Payoffs from sandwich attacks are substantially lower under uniform pricing, for the same trade size, compared to the other two pricing models. For the liquid wETH-USDC pair, sandwich attacks yield higher payoffs under discriminatory pricing, though the difference is in the third digit. For the less liquid wBTC-ETH trading pair, sandwich attack payoffs are much larger under constant product pricing.

To put these numbers into perspective, the last two columns of Table II contain the realized spread that would obtain at the maximal quantity (e.g., 1 and 10 for wETH-USDC, 0.1 and 0.5 for wBTC-USDC). For the liquid ETH-based contract, it is small, less than 2.7 BPS for a trade worth $31K. Such low costs are a sign for a remarkably liquid market, and this spread is comparable to the bid-ask spread on the most liquid centralized exchanges. For the the less liquid wBTC-USDC contract, however, a trade of around $22K would have a realized spread of 800 basis points under traditional pricing and 870 bps with constant product pricing, indicating a meaningful difference between traditional and constant product pricing.

IV. Limiting Sandwich Trades and Order Splitting

A. Technological Solutions

There are several examples of technological solutions that seek to address sandwich attacks. The first is the approach taken by the aggregator protocol 1Inch, which does not allow one to submit opposite-direction trades for some period of time after a trade has been submitted. A sandwich attacker would want the return leg of a sandwich trade to be executed right after the sandwiched trade. Since the attacker has to wait, the profits from the return leg are uncertain, which is a significant disincentive.

The second approach is that taken by COW-Swap ("Coincidence of Wants"), which privately matches large orders, similarly to crossing networks in equity markets. However, COW-Swap essentially just matches orders and thus is closer to a dark pool than an AMM.

A third approach is the Flashbots protocol: validators that follow the protocol receive AMM trades "privately," and they commit to not include them in the public mempool and not to front-run them. Users can connect their wallet to this protocol and when they use an AMM, their trade would be submitted only to miners who follow this protocol. Capponi, Jia, and Wang (2022) document that, since its inception in January 2021, a sizeable fraction of Ethereum miners started to follow this protocol and that around 2% of blockspace on Ethereum is due to transactions from this protocol. However, Capponi, Jia, and Wang (2022) outline that the approach comes with some caveats. First, miners have to voluntarily adopt the protocol and may not have an incentive to do so. Second, when the set of miners is small, users may have to wait for a while before their order gets processed, and they face risk in the meantime. Third, as Capponi, Jia, and Wang (2022) point out, the usage of the flashbots protocol does not eliminate the possibility of a sandwich attack: for the example in the introduction, the sandwiched transaction was sent using the Flashbots protocol.

A fourth approach to curb front-running of trades has been proposed by Aune, O'Hara, and Slama (2017). The idea is to publicly broadcast only a hash of the transaction, not the details of the transaction itself to make it less likely that the transaction can be "attacked." The approach resembles zero-knowledge transactions (which are computationally expensive). This paper was written before swap exchanges and decentralized finance had gained traction, and it is an open question whether this approach is compatible with AMM trades, in particular because the price in AMMs is determined only at the time when the trade is processed in a block.

Another solution is to organize automated market making in a so-called optimistic rollup. A roll-up is similar to a side-chain where users deposit tokens for trading and liquidity provision in a vault, and the roll-up operator processes trades, possibly following a time priority rule. The process differs from a centralized exchange in that the operation of the roll-up uses the functionality and security of the base-layer blockchain, but individual transactions do not enter a public mempool. With a properly designed data layer, the rollup organizer can be incentivized to disallow sandwich attacks with a "slashing" mechanism, i.e., the pool organizer would lose funds if there is evidence of a sandwich attack in the roll-up. One possible downside is that an AMM in an optimistic rollup may not be useful for app-composability, that is, it may not integrate easily in the connected sequences of transactions that occur, e.g., in DeFi loan liquidations (see Lehar and Parlour (2022)).

All the above approaches aim to prevent sandwich attacks entirely. In principle, an AMM could also employ a different pricing rule, such as uniform pricing, which would limit (but not prevent) the profits from sandwich attacks. Uniform pricing, however, incentivizes order splitting. Can technology prevent order splitting? First, validation/gas fees limit the number of profitable splits. Second, similarly to 1Inch, protocols could add a rule that prevents the same wallet address from submitting trades within a specified number of blocks from one another. Doing this, however, would require the system to use data from outside of its smart contract, which increases complexity and may add unforeseen exploit risks.

B. Slippage Protection, Validator Fees, and Trading Fees

Overview. So far this paper has studied price functions without frictions. In practice, liquidity demanders have to pay fees to the blockchain validators and fees to liquidity providers. Furthermore, AMMs allow traders to limit the price impact or "slippage" of their trade so that the trade does not occur if its price exceeds a threshold. In what follows, I study how these feature affect the sandwich attacks.

Limiting Price Impact/Slippage. As the analysis in this paper shows, limiting the price impact of a trade is very important in practice because under constant product pricing, profits from sandwich attacks are unbounded and therefore the liquidity demanders' losses are unbounded, too.

Although limiting slippage may help prevent bad cases of front-running, there are practical concerns. First, liquidity demanders need to understand by how much their trade moves prices because of the shape of the pricing function. If they set a limit that is too low, their trade cannot go through. Second, they also need to account for the possibility that prices can move before validators include their trade, even in the absence of a sandwich attack. By being too cautious, a trader may have to resubmit their trade multiple times.

The problem for the trader is that even non-executed trades can be costly. The reason is that a validator may include an unexecuted trade in a block and collect the gas fees even though no tokens change hands (e.g., the contract fails because the price impact or "slippage" was to large).

Therefore, to avoid paying a gas fee for nothing, a trader may choose to set a slippage limit that is higher than the "natural" price impact that the trade would cause because of the slope of the pricing function. Yet such a higher limit opens them up for a sandwich attack.

Trading and Validation Fees. So far I have ignored trading and validation fees to simplify the theoretical analysis. In practice, users need to pay blockchain validators for the computational cycles (or gas) that the processing of their order consumes. Moreover, in most automated market maker systems, liquidity demanders also pay the liquidity suppliers a fee. Lehar and Parlour (2021) describe the details. Crucially, a sandwich attacker has to pay these fees for both legs of their attack. The main difference between validator and trading fees is that trading fees are exogenous to the protocol, whereas the liquidity demander can choose the validation fee.

C. Slippage Protection and Fees as a Sandwich Attack Defense

In what follows I will discuss how liquidity demanders can use validation fees and slippage-limits to thwart sandwich attacks. Although trading fees also matter in practice, they are not under the control of the trader. As they are merely an additional friction in this decision problem, I will again ignore trading fees to simplify the analysis.

The idea for what follows is that for a fixed level of fees, a trader can specify a slippage limit that makes a sandwich attack unprofitable. The argument also works in reverse: for a fixed slippage level, traders can set a fee that renders attacks unprofitable.

The key insight is this: Because sandwich attackers have to pay the fee twice, liquidity demanders can specify a slippage level that allows for interim price movements (so they don't have to pay for a failed transaction) while still making sandwich attacks unprofitable.

The equations in (12) and (13) specify the sandwich attack profits for the three pricing functions. Suppose a trader specifies a maximum price impact B of their trade so that the trade fails if $P(Q') > B$. A sandwich attacker would then submit sandwich trades with a quantity Q^* that maximizes their payoff without triggering the price impact constraint, e.g.,

$$\max_Q \text{MEV}^{\mathsf{cp}}(Q, Q') \text{ s.t. } P^{\mathsf{cp}}(Q'|X - Q) \leq B.$$

For instance, for constant product and discriminatory pricing, MEV profits are not bounded and increasing in the attack size Q. Therefore, the attacker would simply pick the largest possible Q^*, i.e., $P^{\mathsf{cp}}(Q'|X - Q^*) = B$.

The expressions for the MEVs (12) and (13) do not account for validation fees. In the case of validation fees, the front-runner has to outbid the original trader for the validation fee so as to get the first leg of the attack processed before the original trade. Ceteris paribus, the attacker wants their fee $f + \epsilon_1$ to be as close as possible to the original trader's fee f. Moreover, they also have to pay a validation fee for the second leg of the sandwich attack. The second leg transaction is critical because it provides the positive cash-flows in the sandwich attack. It is in the attacker's interest to place this transaction as close as possible in the block to the sandwiched trade to prevent someone else from earning the positive cashflows. The validation fees for the return transaction therefore have to be high enough to ensure that the second leg of the trade gets included in the same block (otherwise, the front-runner faces a significant execution risk) but they must be low enough so that the attacked trade gets processed before the second leg trade. Therefore, they want to propose a fee close to but below the original trader's fee, $f - \epsilon_2$. The relative sizes of ϵ_1 and ϵ_2 are somewhat arbitrary and without loss of generality, let $\epsilon_1 = \epsilon_2$. Then the attacker pays $f + \epsilon_1 + f - \epsilon_2 = 2f$.

Taken together, there are two new factors that determine the sandwich attack profits beyond the contract liquidity and that are beyond the attacker's control: the price impact or slippage bound and the validation fee. The following result describes how the original trader can prevent sandwich attacks with slippage limits or validation fees.

Proposition 2 (Defensive Fees and Slippage Protection):

1. *Assume that the trader specifies a maximum price impact B for their trade such that $P(Q') \leq B$. Then there exists a validation fee f that the original trader can submit to render sandwich attacks unprofitable.*

2. *Assume that the trader needs to pay a validation fee of f. Then the trader can specify a price impact bound B with $P(Q') \leq B$ such that any sandwich attack is unprofitable.*

Proof. of 1. Fix the original trade size to Q'. For discriminatory and constant product pricing, there are amounts $Q^{\mathsf{cp}}, Q^{\mathsf{dp}}$ that solve $P^{\mathsf{dp}}(Q'|I - Q^{\mathsf{cp}}) = B$ and $P^{\mathsf{cp}}(Q'|X - Q^{\mathsf{cp}}) = B$.

Under uniform pricing, MEV profits are limited; the local maximum for MEV is for $Q = Q'/2$. Therefore, if at the local maximum the slippage bound has not be triggered $P^{\mathsf{up}}(Q'|I - Q'/2) < B$, then the attacker would attack with order size $Q^{\mathsf{up}} = Q'/2$. If at the local maximum for the attacker the slippage bound has been triggered, $P^{\mathsf{up}}(Q'|I - Q'/2) > B$, then because the price $P^{\mathsf{up}}(Q'|I - Q)$ is increasing in Q, the attacker would submit an order $Q^{\mathsf{up}} < Q'/2$ such that $P^{\mathsf{dp}}(Q'|I - Q^{\mathsf{up}}) = B$.

Each of the attack quantities $Q^{\mathsf{cp}}, Q^{\mathsf{dp}}$ and Q^{up} yield the attacker the largest possible MEV given the slippage protection under the respective pricing rules.

Applying these values Q^* to the MEVs in (12) and (13), one can determine the maximum value that the attacker can extract given the slippage protection. For front-running to be profitable,

$$\text{MEV}(Q^*(Q'), Q') > 2f + (\epsilon_1 - \epsilon_2) = 2f. \tag{15}$$

Therefore, for each Q', the liquidity demander can pick a fee f^*, such that the above inequality is violated and then no sandwich attack is profitable.

Proof. of 2. For discriminatory and constant product pricing, MEV is strictly increasing without bound in the size of the attack. Therefore, there exist $Q^{\mathsf{cp}}, Q^{\mathsf{dp}}$ such that $\text{MEV}^{\mathsf{cp}}(Q^{\mathsf{cp}}, Q',) - 2f = 0$ and $\text{MEV}^{\mathsf{dp}}(Q^{\mathsf{dp}}, Q',) - 2f = 0$. By specifying the slippage bounds $B^{\mathsf{cp}}, B^{\mathsf{dp}}$ that solve $P^{\mathsf{dp}}(Q'|I - Q^{\mathsf{dp}}) = B^{\mathsf{dp}}$ and $P^{\mathsf{cp}}(Q'|X - Q^{\mathsf{cp}}) = B^{\mathsf{cp}}$, the trader can render sandwich attacks unprofitable.

In the uniform case if $\text{MEV}^{\mathsf{up}}(Q'/2, Q') - 2f < 0$ then no sandwich attack is profitable. If $\text{MEV}^{\mathsf{up}}(Q'/2, Q') - 2f > 0$, then since $\text{MEV}^{\mathsf{up}}(Q, Q')$ is strictly increasing for $Q \in [0, Q'/2)$ and $\text{MEV}^{\mathsf{up}}(0, Q') = 0$, there is a $Q^{\mathsf{cp}} < Q'/2$ such that $\text{MEV}(Q^{\mathsf{cp}}, Q') - 2f = 0$. By specifying the slippage bound B such that $P^{\mathsf{up}}(Q'|X - Q^{\mathsf{cp}}) = B$, any sandwich attack unprofitable. $\square$

Although liquidity demanders can defend themselves against sandwich attacks, note that for any pricing function, MEV is an increasing function of the original order size Q' so that a larger MEV can be earned for sandwich attacks of larger trades. Therefore, liquidity demanders of large quantities may have to pay ever increasing fees to prevent sandwich attacks.

Moreover, the above mechanism where the liquidity demanders uses a combination of the validation fee and the slippage bound relies on the idea that a front-runner can convince a validator to process their transaction first. However, this argument does not apply if the validator does the sandwich attack themselves: the validator can submit and process their own transaction without paying themselves any fees.

V. Discussion and Conclusion

One of the biggest problems in securities markets is that many assets have an illiquid market. Lack of liquidity makes trading more expensive and increases the risk of not finding a trading counterparty at all, raising fundings costs for issuers and limiting the attractiveness of stock options in compensation packages. A key innovation of automated market makers is the pooling of liquidity, which could improve particularly illiquid markets. A pre-requiste is that the automated trading system is sound. This paper highlights a major design challenge.

By default, while transactions wait to be added to a block, they are visible in public mem-pools. This intrinsic transparency of blockchain operations creates a challenge: an attacker can “sandwich” any trade by submitting a transaction that gets processed before the original one and that the attacker reverses after. The problem is that such an attack is always profitable, modulo fees.

The premise of blockchain-based, automated market makers is that their pricing function is "self-contained:" all information that is necessary for pricing are the deposits in the liquidity as pool, and the pricing rule guarantees liquidity invariance in the sense that as a trader removes liquidity for one token, they must add liquidity in the other to keep aggregate pool liquidity constant. An example for such a liquidity invariance condition is the ubiquitous constant product pricing rule. As this paper shows, any invariance rule renders sandwich attacks profitable, and the attacker's profits are theoretically unbounded.

The problem of profitable sandwich attacks is not limited to invariance rules: An intuitive linear pricing rule that resembles limit order book pricing as the same problem. Another common pricing rule from economic models, so-called uniform pricing, allows only bounded profits from sandwich attacks. However, this rule does not perform well in a blockchain environment either because it encourages order splitting, which lowers the rewards for liquidity providers and increases network congestion.

There are several technological approaches to mitigate the problem: some protocols operate like dark pools, others prevent the return-trip trades central to "sandwiching." Yet another solution is to change the validation process: The Flashbots protocol allows liquidity demanders to send transactions privately to miners, so they do not appear in public mem-pools. Yet in practice, this approach is not a complete solution either. In the future, proof of stake networks or optimistic roll-ups may provide another solution, though as He and Li (2022) shows, punishments for misbehavior are often not imposed.

In practice, there is no "one-system-fits-all" solution. For liquid pools, constant product pricing works well: trades cost about the same or less compared to the calibrated, "economics-based" pricing rules. For illiquid pools, traditional pricing is cheaper. When comes to value lost through sandwich attacks, the uniform pricing rule serves investors best as it conceptually limit profits from sandwich attacks. However, in contrast to the constant product pricing, which is easy to program, the implementation of a uniform pricing rule likely requires a more complex process.

A Appendix: Background on Blockchain Token Trading

Prior to the summer of 2020, most crypto-tokens that had been issued on the various decentralized platforms could only be traded on centralized venues such as Polionex, Binance, Kraken, Coinbase, or the now defunct QuadrigaX. There are subtle institutional differences between centralized venues: some, like Coinbase and Kraken, are connected to the payments network and the traditional financial infrastructure whereas others, such as Binance and Polionex, allow only crypto-to-crypto trades. Transactions on these venues are usually trades of crytocurrencies such as Bitcoin and Ether, the cryptocurrency of the Ethereum network, in exchange for either fiat currencies or other tokens that represent fiat currencies, so called "stablecoins" such as USDT or USDC. To trade, users have to register with the platform and transfer their blockchain asset to the "wallet" of an exchange, before they can use the exchange's system to make their trades. Consequently, as part of the process, custody of the asset moves from the user to the exchange, and users have to pay fees both for the blockchain transaction as well as to the venue. Keeping tokens at an exchange is risky, as demonstrated by the numerous hacking and fraud scandals such as Mt. Gox, QuadrigaCX, or Thodex. Moving crypto assets in and out of traditional exchanges involves fees and is time consuming.

Early in blockchain development, there were a number of projects that operated decentralized limit order books, e.g. EtherDelta. The idea of these decentralized "venues" is simple. Trades are always exchanges of two blockchain tokens, and so users simply register their limit orders as a smart contract on the blockchain, "locking" the token that can be sold subject to someone sending the desired amount of the other token to the contract. When that happens, the contract initiates an atomic swamp: the buyer receives the bought tokens and the seller receives the payment tokens. This atomic swap is processed by the blockchain miners and happens "in one go" so that failures to deliver cannot happen — the trade is the settlement. A decentralized exchange (DEX) provides users with an interface to enter and sign orders using the cryptographic capabilities of their existing blockchain wallet. The system keeps track of these orders (using information from the blockchain, not its own system) and displays available orders on a website.

Although such a mechanism is workable, it requires that individuals continuously monitor and re-submit orders to supply liquidity, a costly activity. Moreover, liquidity provision also involves the ongoing submission of new liquidity providing orders, which absorbs computational power and costs validation fees.[8] Some trading systems circumvent this approach by organizing limit orders off-chain and the system submits only matched orders as a trade to the blockchain for settlement.

[8]This is in contrast to most centralized exchanges or today's stock exchanges, where order submissions are usually free. The exception is Canada where users have to pay a nominal, sub-penny fee for every order that they submit; see Malinova, Park, and Riordan (2013). There are so-called layer-2 systems that keep the limit order book offline and only execute the trades on-chain.

Swap exchanges take yet another approach. A swap exchange is a smart contract that pools deposits of *pairs* of token from numerous liquidity providers.[9] Liquidity seekers interact with this pool of liquidity and offer to add one type of tokens to this contract in exchange for the other token of the pair. The contract then determines the exchanged quantity based on a mechanical rule. Importantly, liquidity providers do not provide a priced contract, they are entirely passive (UniSwap V3 is more general and allows liquidity providers to specify liquidity depending on prices).

REFERENCES

Angeris, Guillermo, and Tarun Chitra, 2021, Improved price oracles: Constant function market makers, Working paper Stanford University https://arxiv.org/abs/2003.10001.

Aoyagi, Jun, and Yuki Ito, 2021, Liquidity implications of constant product market makers, Working paper UC Berkeley https://ssrn.com/abstract=3808755.

Aune, Rune, Maureen O'Hara, and Ouziel Slama, 2017, Footprints on the blockchain: Information leakage in distributed ledgers, Working paper Cornell University https://ssrn.com/abstract=2896803.

Babcock, Bruce, E. Kwan Choi, and Eli Feinerman, 1993, Risk and probability premiums for cara utility functions, *Journal of Agricultural and Resource Economics* 18, 17–24.

Biais, Bruno, 1993, Price formation and equilibrium liquidity in fragmented and centralized markets, *The Journal of Finance* 48, 157–185.

[9]Balancer is more general and can hold portfolios of more than two tokens. If a particular pair is not available as a separate contract, some swap exchanges such as SushiSwap offer cross-contract trading. For instance, suppose someone whats to trade USDC for USDT but there is no such contract. If, however, there is a contract for ETH and USDT and ETH and USDC, then the system arranges a swap from USDC to USDT by trading USDC→ETH→USDT.

Capponi, Agostino, and Ruizhe Jia, 2021, The adoption of blockchain-based decentralized exchanges, working paper Columbia University https://ssrn.com/abstract=3805095.

———, and Ye Wang, 2022, The evolution of blockchain: from lit to dark, working paper Columbia University, Department of Industrial Engineering and Operations Research.

Gans, Joshua S., and Richard Holden, 2022, A solomonic solution to ownership disputes: An application to blockchain front-running, working paper University of Toronto and NBER https://github.com/solomonic-mechanism.

Grossman, Sanford J., and Merton H. Miller, 1988, Liquidity and market structure, *The Journal of Finance* 43, 617–633.

He, Zhigue, and Jiasun Li, 2022, Contract enforcement and decentralized consensus: The case of slashing, working paper https://ssrn.com/abstract=4036000 University of Chicago.

Kyle, Albert S., 1985, Continuous auctions and insider trading, *Econometrica* 53, 1315–1336.

Lehar, Alfred, and Christine Parlour, 2021, Decentralized exchanges, Working paper University of Calgary link.

———, 2022, Systemic fragility in decentralized markets, Working paper UC Berkeley.

Malinova, Katya, Andreas Park, and Ryan Riordan, 2013, Do retail traders suffer from high frequency traders?, Working paper University of Toronto.

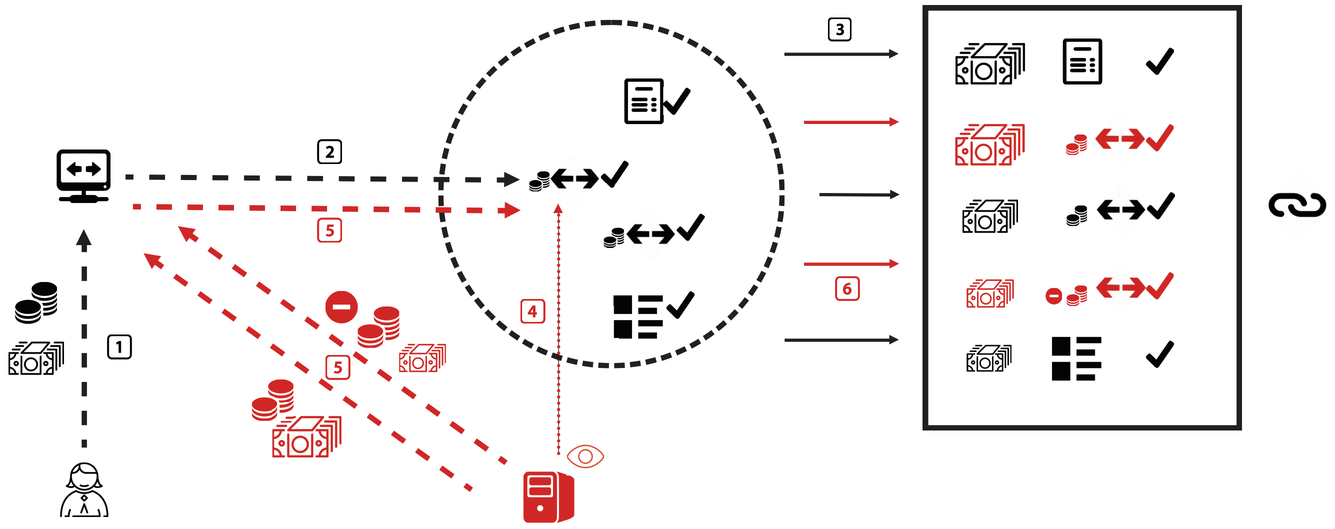

Figure 1
Illustration of Front-Running in Blockchains

The possibility of front-running is an intrinsic feature of blockchains. The schematic works as follows. A user who wants to perform a swap transaction submits the tokens she desires to exchange to the constant product market making contract (1). The contract submits an atomic swap to the blockchain network, and upon verification this transactions enters the mem-pool (2). Verified transactions get ordered in a block based on the fees that they offer (all else equal) (3). An attacker (likely a bot) observes the mempool and see a transaction that can be front-run profitably (4). The bot sends two off-setting swap transactions to the contract, when the front-running trade has a higher fee than the original, front-run transaction (5). In the block, the transactions now get re-ordered according to their mining fees and, upon inclusion on the chain, the transactions in the CPMM contract get executed in the order of the fees (6).

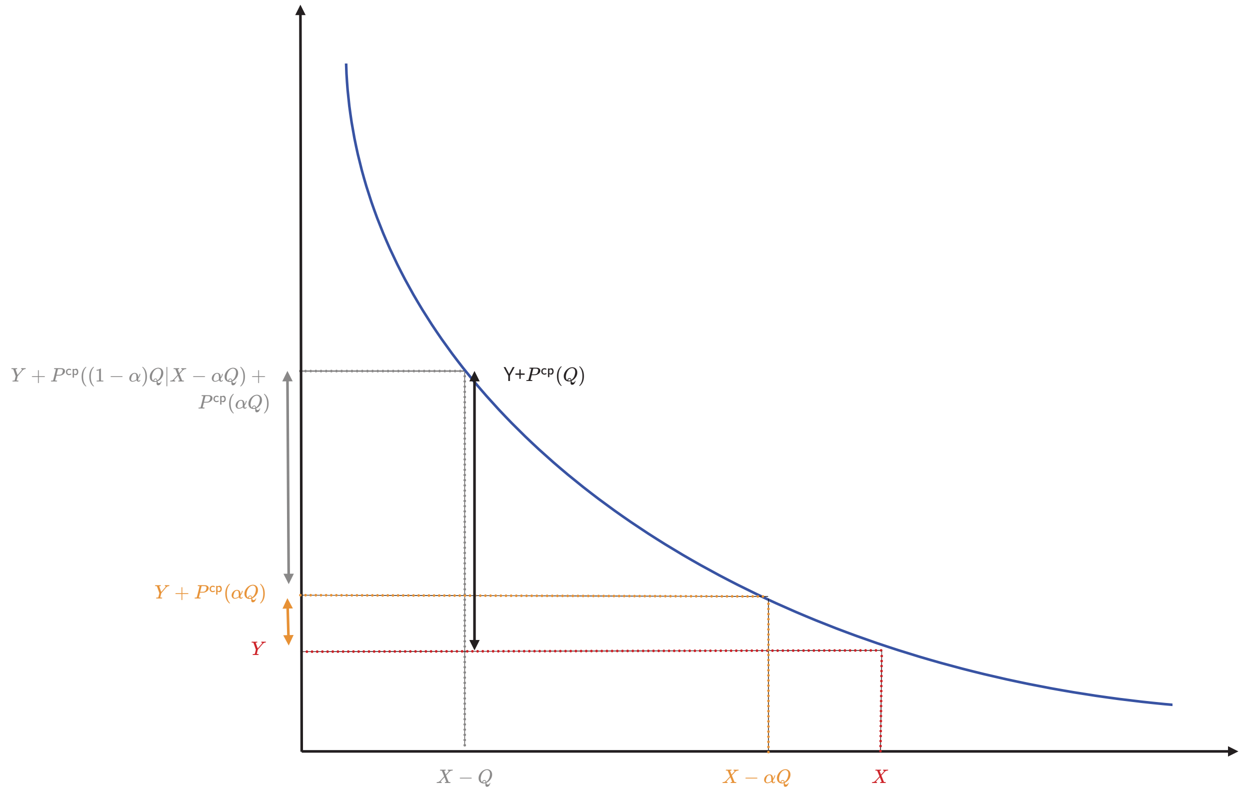
$Y + P^{\mathsf{cp}}((1-\alpha)Q|X-\alpha Q) +$
$P^{\mathsf{cp}}(\alpha Q)$
Y+$P^{\mathsf{cp}}(Q)$
$Y + P^{\mathsf{cp}}(\alpha Q)$
Y
$X-Q$
$X-\alpha Q$
X

Figure 2
Size Consistency for the Constant Product Rule P^{cp}

The figure plots the constant product cost function. After a trade of Q, there are $X - Q$ of the A tokens in the contract and the trader had to pay $P^{\mathsf{cp}}(Q)$ of the B tokens so that there are $Y + P^{\mathsf{cp}}(Q)$ of the B tokens. If instead the liquidity demander had first submitted a trade for αQ of the A tokens followed by $(1-\alpha)Q$, they'd have to pay $P(\alpha Q)$ and $P((1-\alpha)Q|X-\alpha Q)$. By construction, both these prices are on the same iso-liquidity curve and, therefore $P(\alpha Q) + P((1-\alpha)Q|X-\alpha Q) = P(Q)$.

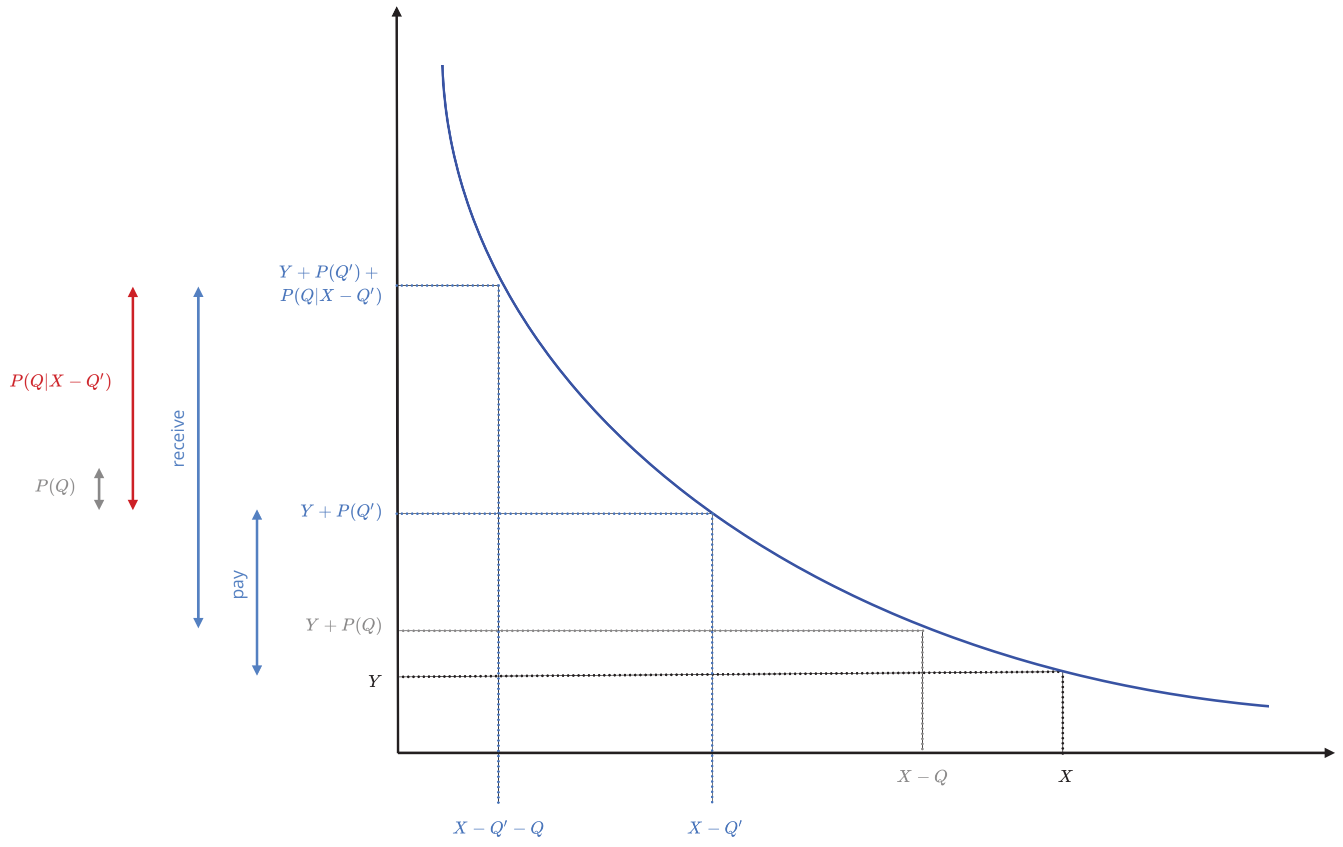

Figure 3
Payoffs from a Sandwich Attack for the Constant Product Rule P^{cp}

The figure illustrates the payoffs from a sandwich attack of an order of size Q. The attack here is for order sizes Q'. The cost is $P^{\mathsf{cp}}(Q')$, the benefit $-P^{\mathsf{cp}}(-Q'|X - Q' - Q)$, the net amount, the MEV, is the red line minus the gray line.

Table I
Calibration of Key Model Parameters

The table summarizes the estimated values for the model parameters for the two trading pairs wETH-USDC and wBTC-USDC. The fundamental value V is based on the respective UniSwap V2 liquidity pools in early 2022 as is liquidity in the pools, X. The standard deviation is estimated from 2021 daily price data, the risk aversion coefficient is from the economic literature, the bound on the quantity distribution is from the traded quantities data as obtained via the blockchain explorer Etherscan. The number of liquidity providers is derived by setting the right hand side of equation (10) equal to $1/N$ and then solving for N.

trading pair	fundamental value V	standard deviation σ	liquidity provided X	bound for quantities L	risk aversion coefficient κ	number of liquidity providers
Source	UniSwap Analytics	Coindesk	UniSwap Analytics	Etherscan	Literature	solve (10)
wETH-USDC	\$3,100	260	38,100	10	0.0005	416
				1	0.0005	409
wBTC-USDC	\$44,000	4100	6.25	0.5	0.0005	1.2
				0.1	0.0005	1.2

Table II
Cost Comparison

The table summarizes the estimated costs for traditional vs. constant product pricing. The average price is computed for the quantity distribution that underlies the respective model. Ex ante expected prices are theoretically identical for uniform and discriminatory pricing for zero starting inventories and are therefore not differentiated.

	average price paid		Maximum Extractable Value for $Q' = Q$			Maximum Extractable Value for $Q' = Q/2$			realized spread in BPS of V for $Q = L$	
trading pair	uniform & discr.	constant product	discrim-natory	uniform	constant product	discrimi-natory	uniform	constant product	uniform & discr.	constant product
wETH	3100.407	3100.407	2.715	0.000	2.714	1.357	0.339	1.356	2.637	2.625
-USDC	3100.041	3100.041	0.028	0.000	0.020	0.014	0.003	0.010	0.267	0.262
wBTC	45751.040	45859.880	583.680	0.000	710.661	291.840	72.960	337.992	795.928	869.565
-USDC	44350.210	44355.800	23.347	0.000	24.337	11.674	2.918	12.058	159.187	162.602