# Feedlot: trading with safe, low latency price oracles [*]

*The Feedlot working group*

> *feed* — Something supplied continuously.
>
> *lot* — One or more items auctioned or sold as a unit, separate from other items.

**Abstract**  This article is a feasibility study for a class of automated market makers (AMMs), collectively dubbed *Feedlot*, that passively provide liquidity at a price based on the uniform clearing price (UCP) of a competitive batch auction such as CoW protocol. If implemented properly, liquidity providers on Feedlot AMMs should enjoy protection from the adverse selection that plagues CFMM LPs, and oracle fees could provide a new revenue stream for the batch auction provider itself.

We review general principles of designing CFMMs and pricing oracles; propose an architecture for a channel through which the CoW UCP is communicated to a liquidity pool; and review the security implications for CoW and the pool of consuming the UCP to execute additional trades. We study from theoretical and empirical perspectives the economic implications for traders and LPs that a functioning Feedlot implementation would entail.

This article is not a whitepaper, specification, or fully fleshed out proposal, and it makes no claim to exhaustiveness or completeness on any of the topics discussed — particularly security. Would-be implementers of a Feedlot AMM are advised to pursue thorough further investigations of these issues in the context into which they hope to deploy.

## 1   Introduction

The most widely used type of blockchain market today is the constant function automated market maker (CFMM), whose deployments collectively process trading volume to the value of tens of billions USD per month [28, 1, 10]. A CFMM provides an environment where anyone can deploy funds and passively accept trades in exchange for trading fees. Price impact is calculated in terms of the reserves using highly space and time efficient algorithms.

The popularity of the CFMM model has brought to light some basic difficulties. In addition to fundamental risk, liquidity providers on CFMMs are exposed to the effects of **adverse selection**: they incur losses from being forced to accept trades with informed actors at unfavourable prices. Many groups of liquidity providers consistently lose money on their investment [17]. Meanwhile, the permissionless and public environment in which CFMMs execute exposes naïve traders to the activities of MEV bots, who take advantage of the trader's revealed intentions together with the market's deterministic price impact function to manipulate prices and extract value from a pending transaction [30, 13].

Various schemes have been proposed to address some of the issues associated with fully automated markets [8, 14, 20, 9]. Some aim to protect users by providing a dedicated channel that provides additional privacy, failure, or ordering guarantees. Among these, blockchain implementations of the *RFQ batch auction model* socialise pricing and protection from ordering attacks by settling trades atomically in batches with a *uniform clearing price* (UCP). This price is provided by the winner of a competition to find routes for orders that maximise the welfare of their creators.

Another class of proposals adopt the idea of adjusting pricing using an external price feed [27, 26]. If this price feed is sufficiently reliable, passive liquidity providers can mitigate or even entirely avoid losses to adverse selection. However, employing such a price feed for trading provides a strong incentive to deliberately skew, interrupt, or otherwise manipulate the published price. A typical price feed pipeline has several phases that could fail — by accident or design — to preserve the

---

integrity of the published data. Examples of cases where financial losses have occurred as a result of relying on such feeds are plentiful [16, 4, 23, 7].

In this report, we discuss a proposal to use the UCP provided by an RFQ batch auction with on-chain settlement — specifically, CoW protocol[1] — to adjust the pricing of a liquidity pool. The fact that the UCP is used to settle a batch auction means that it has a unique guarantee not shared by other price oracle data sources: it is *realisable* in the sense that it comes with an on-chain proof that it could actually be used to execute trades (at least, before the batch).[2]

We give the working name *Feedlot* to a hypothetical AMM based on this model.

The rest of this report is structured as follows: in §2, we review general principles of designing CFMMs and pricing oracles; in §3 we discuss the channel through which the UCP is communicated to a Feedlot liquidity pool; in §4 we review the security implications for CoW and Feedlot of consuming the UCP to execute additional trades; finally, in §5 we study from theoretical and empirical perspectives the economic implications for traders and LPs that a functioning Feedlot implementation would entail.

# 2   General design principles

## 2.1   Definitions

Our blockchain model is based on Ethereum, and should apply to any blockchain-based state machine with similar principles (for example Gnosis Chain). By a *smart contract system* we mean any kind of on-chain entity or aggregation of entities. In particular, it may mean a single smart contract (e.g. a Uniswap pool) or a structured collection of interacting smart contracts (e.g. Uniswap as a whole).

We make use of the following standard terminology:

- *Liquidity pool.* Smart contract system that custodies the assets of *liquidity providers* and tracks withdrawal liabilities.

- *Liquidity provider (LP).* An agent that deposits funds in a liquidity pool.

- *Automated market maker (AMM).* Smart contract system that provides pricing and passively settles orders by trading against a liquidity pool.

- *Batch auction.* Order settlement system that accumulates orders in a buffer and settles them with a uniform clearing price. Prices are supplied by *solvers* who compete to optimise an objective function defined in terms of the price vector and the set of orders.

- *Uniform clearing price (UCP).* Price vector against which a batch auction is settled.

- *Coincidence of wants (COW).* Execution or partial execution against one another of two orders in opposing directions.

- *Solver.* Agent that provides quotes in competition to settle a batch auction.

- *Constant function market maker.* An AMM whose pricing function depends only on reserves.

For simplicity, we will consider only liquidity pools with two assets $A$ and $B$ whose balances are denoted $(x, y) \in [0, \infty)^2$. We take token $A$ for the numéraire (also called the 'quote token'), so that prices are for token $B$ in terms of token $A$. If a trade occurs of $\Delta x$ $A$ tokens for $\Delta y$ $B$ tokens, the execution price is $\Delta x / \Delta y$.

## 2.2   Constant function market makers

A constant function market maker (CFMM) is defined by its *invariant function* $f(x, y)$, a real-valued function of the pool reserves. The associated marginal pricing function is

$$p(x, y) = \frac{f_x(x, y)}{f_y(x, y)}.$$

If reserves are in state $(x_0, y_0)$, the amount of $B$ tokens paid out in exchange for $-\Delta x$ $A$ tokens is

$$\Delta y = \int_{x_0}^{\Delta x} 1/p(x, y(x)) \, dx - y_0$$

where $y(x)$ is determined by the corresponding differential equation $y'(x) = 1/p(x, y(x))$ and the initial conditions $y(x_0) = y_0$.

A practical implementation of market orders on a CFMM must have a computationally efficient algorithm for computing $\Delta y$ given $\Delta x$. On the other hand, to decide if a given swap $(\Delta x, \Delta y)$ will be accepted by a CFMM, it is enough to compute the invariant $f(x + \Delta x, y + \Delta y)$.

---

[1] https://docs.cow.fi
[2] In the language of [25], it is backed by *trade collateral*.

## 2.3   Liquidity provider costs

The CFMM quote is not automatically updated when changes occur on other markets. Rather, price updates are supplied by arbitrageurs who trade on the CFMM in such a way as to push the pricing into alignment with external markets.

From a dual perspective, this trading activity has the effect of adjusting the balance of assets in the pool towards the value-minimising point on the level curve, computed in terms of external market prices. For example, with Uniswap's constant product formula $f(x, y) = xy$, the equilibrium balance is 50-50, that is, $px = y$.

LPs effectively pay for this adjustment service by trading at unfavourable prices [18]. The amount they pay depends only on the external price movement and the second derivative of the optimal pool value function. In particular, there is no room for price updaters to compete with one another on the basis of reducing cost to LPs; rather, all arbitrageur competition is at the infrastructure layer and excess rewards tend to be absorbed either by consensus nodes or by middleware services that assemble blocks or partial blocks [5]. LPs cannot concentrate liquidity near a single price without exposing themselves to more severe adverse selection impact.

## 2.4   Automated market makers with a price feed

The issue of uncontrolled LP losses to adverse selection could be alleviated if price updates from external markets could be supplied by another channel, so that the marginal pricing function could depend on inputs other than reserves:

$$p = p(x, y; p_{\text{ext}}).$$

In blockchain applications, such channels are called *price oracles* [12, 19, 3].

*Example.* The most obvious choice for an oracle-based pricing function is

$$p(x, y; p_{\text{ext}}) = p_{\text{ext}},$$

so that the AMM simply quotes the oracle price. We refer to this as *passthrough* pricing. If we black-box the data source and assume that it provides no-arbitrage prices, then such an AMM would indeed be protected from adverse selection.

However, as is well-known, oracles in the field are fallible. They are vulnerable to bugs and deliberate attacks at each stage of the data pipeline. Oracles that provide prices that anyone can trade against are a particularly attractive vector for sabotage, both because of the potential for financial gain and because, in the case of a manipulated data source, of the difficulty in establishing that an attack has even occurred [24, 25].

Moreover, simply passing through the oracle price offers no control over inventory management. In particular, if trading tends to occur more on one side than the other for whatever reason, reserves of the in-demand asset could easily dry out [11, §2.2]. In traditional finance, this risk is typically accounted for by adjusting pricing to be more or less favourable depending on whether a trade would push reserves towards or away from the desired inventory [2]. Assuming a typical microeconomic environment with negatively sloped demand curves, order flow naturally skews towards the buy (resp. sell) side as prices edge below (resp. above) the market clearing price. If prices slide far enough for arbitrages to appear, one can further suppose that a population of informed traders will rapidly effect the desired inventory adjustments.

*Example.* Suppose we target a reserve balance of 50-50 (computed in terms of the oracle price), setting

$$p(x, y; p_{\text{ext}}) = p_{\text{ext}} \cdot g\left(x / y p_{\text{ext}}\right)$$

where $g : (0, \infty) \to (0, \infty)$ is a monotone increasing function such that $g(1) = 1$. The most basic choice is $g(r) = r$, in which case the price oracle drops out of the formula and we recover the UniswapV2 pricing function $p(x, y; p_{\text{ext}}) = x / y$.

*Example.* More interesting is to choose a function which has a small or vanishing gradient near 1. A simple example is

$$g(r) = \begin{cases} r\kappa & r \leq \kappa^{-1} \\ 1 & \kappa^{-1} \leq t \leq \kappa \\ r\kappa^{-1} & \kappa \leq r \end{cases}$$

where $\kappa > 1$ is the 'tolerance ratio' of the pool for reserves to deviate from an even balance. An AMM trading on this pricing function provides passthrough pricing as long as the inventory balance remains in the range $[\kappa^{-1}, \kappa]$; a simple calculation shows that outside that range it trades as a weighted CPMM with weights $(1, \kappa)$, resp. $(\kappa, 1)$, depending on whether we have too little numéraire or too much, respectively.

More generally, 'stableswap' curves used in the wild for liquidity pools with weakly pegged prices can be adapted using this price feed approach to trade volatile assets as well [6, 22].

*Example.* Yet another approach is to provide CPMM pricing, but bound execution prices by the oracle price in the direction unfavourable to the LP. This is similar to the pricing function approach in that the CPMM preference curve discourages trading that would push the inventory too far from the desired balance, but the normalisation of this curve updates only when the AMM accepts a trade, rather than when the oracle updates. The hybrid model has the benefit of ease of implementation.
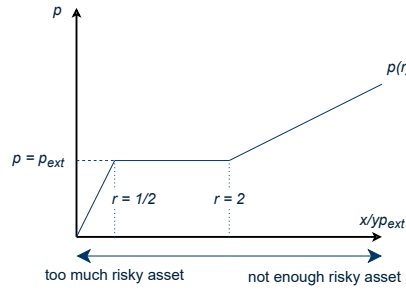
Figure 1: Piecewise linear pricing function of reserve balance with tolerance ratio $\kappa = 2$.

## 2.5    Design goals

A successful implementation of a Feedlot AMM should have the following properties:

- Liquidity providers should enjoy cheap portfolio management, some yield, and protection from adverse selection.

- Traders should enjoy low, predictable fees, control over execution time, and favourable prices at least for trades in the 'correct' direction.

- It should not be economical to manipulate the uniform clearing price (UCP) of the CoW batch auction in order to trade at favourable prices on Feedlot.

  Moreover, it should not be possible to substantially offset the cost of manipulating the CoW UCP (for any reason) by trading on a Feedlot AMM.

- Wherever possible, Feedlot should use incentive-compatible mechanisms to ensure correct operation. Social adjudication procedures should be employed only as a fallback.

In subsequent sections, we discuss approaches to implementing an AMM realising this list.

# 3    Architecture

Our model for a batch auction system is the CoW protocol on Ethereum. The CoW protocol consists of a set of off-chain entities that we collectively refer to as *CoW services* and a system of Ethereum smart contracts called the *CoW contracts*. The details of the internal architecture of these aggregations is not discussed here.

Schematically, the CoW algorithm runs as follows:

1. Traders send orders to an off-chain order book server where they are tracked in a database.

2. A set of offchain entities called *solvers* query the database and attempt to construct a *solution*, that is, an Ethereum transaction that collectively settles all user orders at a fixed price vector $\vec{p}$ — the uniform clearing price. This transaction may contain arbitrary Ethereum message calls.

3. Every batch interval, solutions are validated by simulating against a recently observed chain state and ranked according to a utility function. The utility function measures the total marginal utility of user orders filled by the solution in terms of the difference between the limit price and the fill price. Utilities of orders denominated in different tokens are normalised using prices on whitelisted external markets.

4. The solver with the winning solution calls the `settle()` function on the CoW settlement contract, which executes their solution, emitting `Trade` events.

5. Misbehaviour is assessed socially. Slashing punishments can be triggered by a vote of the CoW DAO.

An AMM whose quote depends on the UCP of the CoW batch auction needs to have at least the following structures:

- A way for traders to submit orders.

- A way for would-be LPs to add or remove liquidity.

- A communication channel $\mathcal{O}$ connecting Feedlot with the CoW contracts or services along which can be communicated the UCP.

The first two components are standard elements of AMM implementation. The last item is the distinguishing feature of Feedlot AMMs, so in this section we focus on the design of this channel.
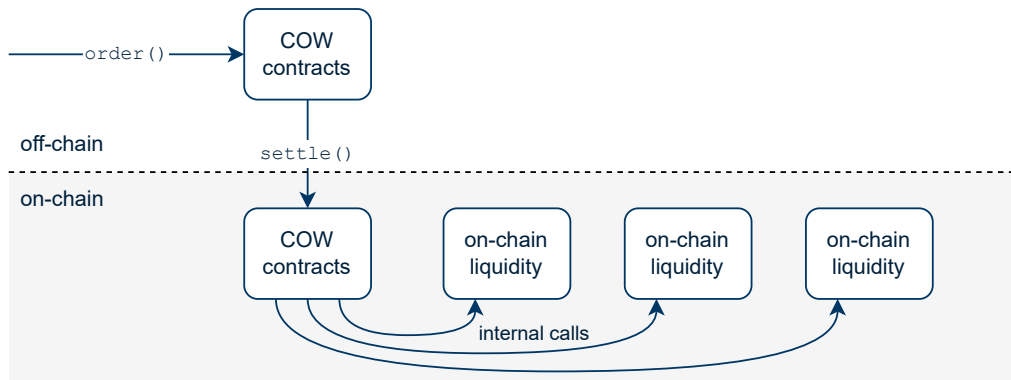
Figure 2: CoW procotol execution. CoW services accumulate orders and batches are settled (at most) every 30 seconds.
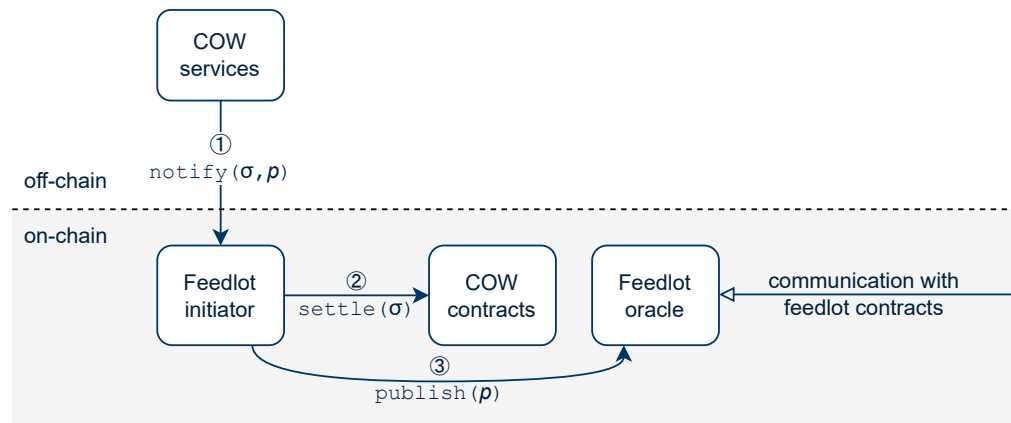


Figure 3: Atomic execution of the batch solution and publication of the UCP. The ordering of (2) and (3) can be reversed.

## 3.1   Synchronisation

A batch auction system needn't be synchronised with the blockchain. There may not be an auction every block, and even if there is, not every token pair traded on the BAS need necessarily appear in the batch. This is the case for CoW: auctions are roughly every 30 seconds, and sometimes no auction occurs for hours at a time. Therefore, a fresh batch auction price might not be available at the point that a trader wishes to trade on feedlot.

  If the Feedlot quote price is to depend on the price of a CoW batch auction, then, there are only two possibilities:

1. The quote function uses a stale CoW price (poll model).

2. The market only settles trades at the same time as the batch auction (subscription model).

These two options are analogous to what in the blockchain oracles literature have been called the *pull* and *push* models, respectively [12, 19].

  In either case, CoW services can guarantee zero latency updates of the UCP by writing into the channel atomically together with the batch execution transaction. This would require minor modifications to CoW's offchain components so that they call a wrapper contract that both triggers the CoW settlement and publishes a price to the on-chain oracle. It does not require any changes to CoW's onchain components.

  What could be considered a third option is to fall back on another algorithm in case no price is available. Since that essentially amounts to not trading on Feedlot, and could easily be implemented at a higher layer, we don't discuss that here.

### 3.1.1   Poll model

In this model, CoW writes the UCP vector to the channel every time there is a batch. Meanwhile, Feedlot AMMs query the price whenever they have to settle an order. There is no synchronisation between these two tasks. One write is needed
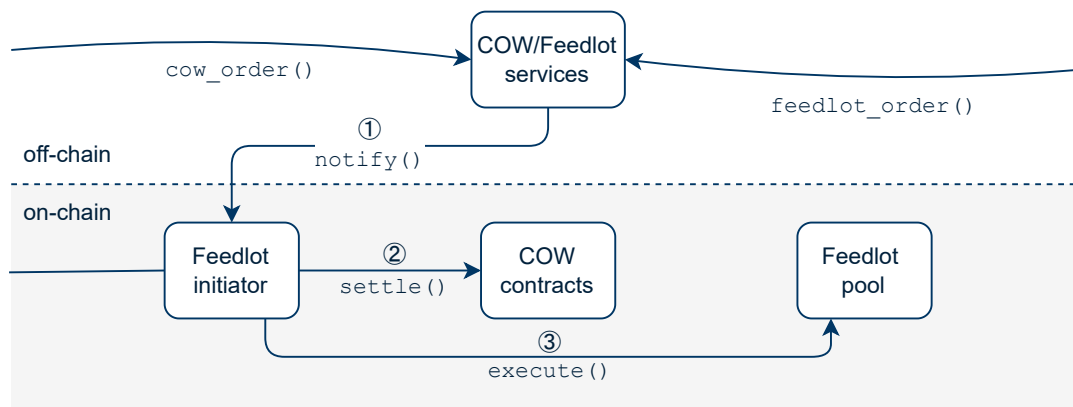
Figure 4: Integrated oracle with offchain order queue. Feedlot AMM executions are triggered when a new price is published.

per batch per product, and one read is needed per trade on Feedlot. These operations being relatively cheap, it would be reasonable to implement this type of channel as an on-chain oracle. A variant approach is to aggregate UCPs to produce, for example, time-weighted average prices. This is the approach taken by Uniswap's oracle service [1].

Since the prices retrieved by this method are necessarily stale, this approach does not preserves the unique *realisability* property of the UCP (see §4.1). For this reason, and because poll-based oracle services are already plentiful, for the rest of this report we focus instead on the more interesting subscription model.

### 3.1.2   Subscription model

In this approach, orders arriving at Feedlot are accumulated in an order book. All trades are executed when there is a CoW batch. Whether the Feedlot trades execute before or after the CoW batch affects only how Feedlot behaves for orders placed in the winning solution itself; see §3.2. Orders that arrive at Feedlot outside the batch are not affected by this ordering.

As with any order book based protocol, the order book itself would potentially be quite expensive to maintain on the Ethereum chain. It is beyond the scope of this report to discuss approaches for optimising trust model and cost-effectiveness of off-chain order books. Pragmatically, a reasonable approach would be to maintain the order book in the same place as the order book for the CoW batch auction, which at time of writing is a set of servers maintained by the CoW team.

## 3.2   Can the batch itself settle on Feedlot?

Certain configurations would allow the solution itself to both settle orders on Feedlot and quote it a price. Evidently, this can only happen if Feedlot orders are executed *after* execution of the solution but *before* verifying that the solution satisfied all the orders it claimed to satisfy. We call this model *CoW native*, because in this case only the CoW batch solution can interact directly with the AMM; here Feedlot acts as a kind of private liquidity pool for solvers. Note also that in this case, AMM orders do not have to wait in a queue to be executed (though of course, the user orders being settled by the batch had to wait for the batch as usual).

Implemented naïvely, this design permits the winning solver to quote any price at all and have it instantly execute against Feedlot. Since the solver's objective is to get the most favourable price possible for its users, a rational solver will quote the least favourable price that Feedlot will accept. Naturally, allowing an entity to trade against the AMM at any price it chooses is trivially exploitable to the detriment of LPs, and so special care has to be taken to address this case. While the ability for solvers to both set prices and trade at those prices on Feedlot persists even when the batch itself cannot settle on Feedlot — see §4 — this design is the only situation in which doing so is completely risk-free for the solver.

Nonetheless, some of the mitigation strategies discussed in §4.3 still apply to this case. By adjusting the solver objective function and by carefully defining and monitoring solver activities, it may be possible to make a CoW native AMM that protects its LPs from adverse selection while providing a safe execution environment for solvers.

## 4   Cryptoeconomic security

Assuming correct implementation, a generic oracle $\mathcal{O}$ can fail in two ways:

- Inaccurate or unreliable data source.

- Failure to faithfully communicate the value from the data source.

In the present context, fidelity of the communication of value is guaranteed by non-falsifiability of on-chain execution. In this section, we address the reliability of the CoW UCP as a data source in two senses: *realisability* and *manipulation resistance*.

## 4.1  Realisability of pricing data

We say that the price $p$ is *realisable* for an agent $I$ at a volume $V$ of the in token and chain state $\phi$ if it is possible for $I$ to buy $V$ $A$ tokens for $pV$ $B$ tokens in state $\phi$. This is a way to make sense of the 'truthfulness' property for oracles sketched in [12].

   If CoW solver prices are not realisable, the solver must settle the trade with their own private liquidity held in bond by CoW DAO. A posteriori, CoW solver prices are realisable for the solver at the state immediately before the batch and at the volume traded in the batch. That is, the structure of the CoW batch auction itself guarantees that the UCP vector is realisable at the volume traded in the batch.

   It is worth noting that merely simulating executions on-chain (and reverting all changes at the end) is sufficient to verify realisability of the price vector.

## 4.2  Manipulation resistance

Wherever economic decisions are made based on the output of an oracle $\mathcal{O}$, there lies an incentive for individual or cooperating actors to deliberately effect changes in the value published by $\mathcal{O}$ in order to influence those decisions. In few settings is this incentive more transparent than the present case of a price oracle coupled with an automated market that offers to trade at or near that price. If, for example, some party is able to exert control over $\mathcal{O}$ at a cost $C_{\text{manip}}$ so that it publishes a price $p$ that is strictly less than a price $p_{\text{ext}}$ realisable on some other market, then a strategy of doing so and then buying a quantity $V$ of the risky asset on Feedlot yields a profit of

$$V \cdot (p_{\text{ext}} - p) - C_{\text{manip}}. \tag{1}$$

Even if (1) is negative, trading on Feedlot in this way *offsets* the cost of manipulation, which may still be strategically optimal because of other decisions contingent on the output of $\mathcal{O}$. Note that this type of activity does not undermine the fidelity of the oracle itself: a manipulated price may still be realisable. However, it does constitute a centralising force, and moreover, one that acts in direct opposition to LPs, presumably driving away liquidity.

   If a Feedlot implementation is to succeed, it must therefore be **manipulation resistant** in the sense that the cost of effecting a change in the oracle price away from its equilibrium value — more accurately, what would be its equilibrium value if Feedlot did not exist — exceeds the marginal proceeds of manipulation. To analyse this property concretely, we need to understand what strategies are available to a manipulating adversary. We enumerate a few basic examples here:

*Example* (Price setting through private liquidity). Volume $V$ of buy orders is sitting on CoW for a given pair with external market price $p_{\text{ext}}$. A solver submits a solution in which they route private liquidity to settle the orders at price $p < p_{\text{ext}}$; if their solution wins, they effectively lose $V \cdot (p_{\text{ext}} - p)$. At the same time, they enqueue buy orders on Feedlot of volume $V'$.

   If there are few orders in the opposite direction, the solver has a good chance of winning the auction because they provided a market-beating price. The solver can even cheaply inflate this chance by adding his own buy orders to CoW (i.e. wash trading). In this case, their orders on Feedlot execute at price $p$, and the solver nets a profit of $(V' - V) \cdot (p_{\text{ext}} - p)$.

*Example* (Price setting by arbitraging a CFMM). Suppose we are in the same setting as the above, but Feedlot is configured so as to accept pricing only from whitelisted CFMMs. A trader with sufficient capital can manipulate the price on a CFMM so that it becomes possible to buy volume $V$ at price $p$, and manipulate it back afterwards. If successful, the effective cost of this strategy is again $V \cdot (p_{\text{ext}} - p)$ (less fees). With atomic execution in the batch, additional capital requirements can be financed with a flashloan.

*Example* (Permissionless price setting by arbitraging a CFMM). Any trader can attempt the CFMM manipulation attack, and solvers would be incentivised to source liquidity from their manipulated markets. However, in this case the trader cannot guarantee atomic execution, so he cannot use flashloans and faces significant execution risk. In particular, a rational solver could arbitrage the CFMM back into equilibrium after settling user orders and before the back end of the trader's sandwich.

   In general, disregarding fees and execution risk, we argue that the cost of manipulation is essentially the cost of trading at the unfavourable manipulated price with entities not collaborating with the adversary. That is, it becomes $V \cdot (p - p_{\text{ext}})$ where $V$ is the volume of trades in the batch submitted by agents other than the adversary, $p_{\text{ext}}$ is the external market price, and $p$ is the manipulated price. Hence, risk-free price manipulation is profitable exactly when $V' > V$. More generally, trading on Feedlot subsidises the manipulation by an amount depending on the 'real' volume on CoW.

## 4.3  Mitigating the manipulation problem

Price manipulation is a fundamental threat to Feedlot LPs and centralisation vector for CoW protocol itself. We do not have a complete solution for this problem, and any team wishing to push forward on the Feedlot paradigm must invest resources into developing a mitigation strategy.

   While there is probably no 'magic bullet' design concept that makes all manipulation strategies unprofitable, we can identify several promising general approaches, some combination of which may yield a workable design:

- *Make manipulation more expensive.* Discounting wash trading, the fundamental cost to make a manipulated price $p$ realisable at a volume $V$ is $V \cdot (p - p_{\text{ext}})$. However, the capital requirements and risk associated with doing this depend on the details of the market on which this price is realisable.

  Feedlot should only accept prices sourced from whitelisted markets where manipulation is capital-intensive — for example, deep CFMM pools — and should be structured so that such manipulation cannot be carried out atomically.

- *Constrain the marginal proceeds of manipulation.* Carefully implemented, wash trading aware volume controls could prevent the adversary from turning a profit even if he is able to influence the CoW price by dominating volume. Volume limits could be expressed in terms of CoW volume (less wash trading) and absolute parameters set by DAO governance.

- *Control solver incentives.* Solvers, as the most privileged actors in the CoW ecosystem, are also the most powerful threat to Feedlot LPs. On the other hand, if their incentives were properly aligned, they could also be a powerful defense against manipulation — for example, by arbitraging manipulated markets back into ex-Feedlot equilibrium before settling. To accommodate Feedlot, the objective function of solvers should be adjusted to strike a balance between utility for user orders and for Feedlot LPs.

- *Define, expose, and punish solver misbehaviour* Solvers who manipulate prices to the detriment of Feedlot LPs can, if this behaviour is well-defined and observable, be slashed by the DAO. Pinning down the notion of 'manipulation' in general is not straightforward [15, 29], but in the present context a plausible definition is *behaviour that would be irrational, less an effect on price* [21]. For rationality to be observable and decidable, we must develop practical models for solver incentives, and solvers should declare their conflicts of interest.

# 5   Economic impact

## 5.1   For LPs

In §2.4, we characterised the introduction of a price feed to our AMM pricing function as a way to mitigate or prevent LP losses due to adverse selection. Can we quantify the effects of this mitigation in practice?

In [18], the authors introduce a framework to quantify losses of CFMM LPs to adverse selection that depends on an external market price $P$. Adapting this framework to discrete time and with the CoW UCP playing the rôle of the external market price, we can define the *CoW-based loss-versus-rebalancing* as

$$\mathrm{LVR}_n := R(P_n) - V(P_n)$$

where:

- $P_n$ is the UCP of the $n$th CoW batch;

- $V(P_n)$ is the optimal CFMM pool value at an external market price of $P_n$;

- $R(P_n) = V_0 + \sum_{i=1}^{n} y^*(P_n) \cdot (P_n - P_{n-1})$ is the value of the self-financing rebalancing portfolio at the time of the $n$th CoW batch, where rebalances take place only at CoW batch times. (If trading fees are negligible, path-independence means that it doesn't actually matter when rebalancing takes place.)

Discrete-time analogues of the arguments of *op. cit.* show that this is a non-negative and non-decreasing process.[3] In continuous time, the quantity can be expressed in terms of the instantaneous square volatility $\sigma^2(P)$ and $V''(P) > 0$; one can derive similar, though uglier, formulas for the discrete case.

If Feedlot uses passthrough pricing and accepts trades in such a way that its reserves track the rebalancing portfolio, then $R_n$ can also be interpreted as the portfolio value of a Feedlot LP. That is, $\mathrm{LVR}_n$ is the difference in performance between a Feedlot LP and a CFMM LP. This construction is contingent on a sufficient 'uninformed' order flow arriving at Feedlot for it to track the reference portfolio by selectively accepting trades.

**Simulations.**   We simulated the portfolio metrics of LPs for a simple Feedlot implementation which rejects trades that would push inventory balance (in the sense discussed in §2.4) outside an interval $[\kappa^{-1}, \kappa]$, where $\kappa > 1$. We also compared this with the hybrid model that uses a CPMM preference curve. We used three types of data source for order flow:

1. Trades on CoW;

2. Trades on UniswapV3;

3. Randomly generated 'zero-intelligence' order flow (Poisson arrivals, uniform direction, exponential size).

---

[3]We lose the predictability property, because of course, the discrete-time analogue of a diffusion process is not predictable.
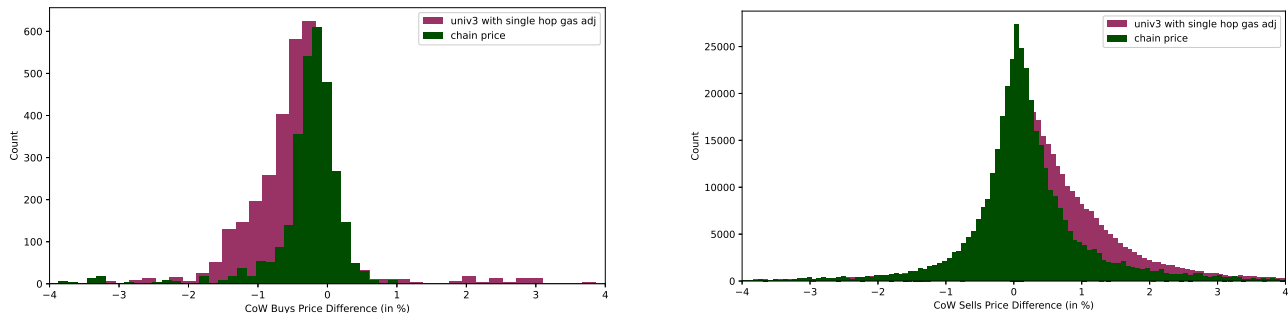
Figure 5: WETH/USDC price deviation on CoW, aggregated from September 2021 to March 2023 and separated into buy and sell side for CoW and Univ3. Note that sell side had much larger volume over the period due to general market conditions.

The trades datasets used for simulations 1 and 2 are the same ones used in §6; see there for details. The datasets don't record whether the trades arose from buy or sell orders, or what the limits were, so for simplicity we made the ansatz that all orders were market sell orders.[4] For comparison, we also simulated the same order flow against a CPMM.

[DATA GOES HERE]

## 5.2   For traders

Fundamentally, a trader on Feedlot makes a commitment to buy a product at the price set by a certain oracle at the next oracle update (volume limits notwithstanding). By construction, this price is also realisable on other markets; that is, this opportunity already exists 'naturally.'[5] Why then should traders be interested in trading on Feedlot?

We identified three legitimate reasons:

- *Protection.* By trading along with the CoW batch, the risk of price changes between commitment and execution time (whether due to 'natural' variability or deliberate attack) is socialised across the whole CoW/Feedlot execution. Note that traders on CoW itself also enjoy this benefit.

- *Spread.* User orders on the CoW batch auction employ solvers to actively seek out favourable prices. On the other hand, market orders on Feedlot simply wait for someone on CoW to ask for a price for that pair and then take advantage of the result. Feedlot LPs do no price-finding work, and depending on how the liquidity curve is configured, may also enjoy portfolio management services (at the cost of unpredictable execution prices for users). Hence, Feedlot trading fees (or equivalently, the bid-ask spread) must be cheaper than the active service provided by CoW protocol.

- *Skew.* Depending on the design of the liquidity curve, a Feedlot AMM may provide more favourable pricing for trades that push the pool reserves towards the desired balance. From the trader's perspective, this is a similar dynamic to the way prices are updated on a CFMM.

## 6   How good a price oracle is CoW?

The structure of Feedlot implicitly assumes that the CoW UCP is a kind of non-directional market midrate to which we can add spread in the form of trading fees. However, in practice, most volume on CoW settles on AMMs and so has some spread already baked into it. How much error does this spread introduce, and how does CoW measure up against other non-directional pricing data feeds?

To study this, we obtained trade data for WETH/USDC orders on CoW, Uniswap v3 USDC/WETH 5bps and 30 bps fee pool swaps, and ETH/USD Chainlink oracle price data from August 2021 to March 2023. We made an adjustment to the Univ3 spot prices to account for gas transaction costs. Since we only had access to the gas limit, which is an upper bound for the amount of gas used in a Univ3 transaction, we used the median gas limit (around 312,218 gas) as an heuristic to calculate the transaction cost. For reference, a single Univ3 swap transaction starts at around 130,000 gas. Thus we can interpret the median value as assuming the average Univ3 swap goes through a little more than 2 hops on average.

With the median gas adjustments, traders received a 1.2% discount to buy and sell WETH through CoW. With the single hop gas adjustments, which serve as a hard lower bound, traders received a .43% discount to buy and sell WETH.

---

[4]The advantage of using sell orders over buy orders is that it automatically guarantees finiteness of the traders' own inventories, which would otherwise have to be managed to avoid infinite price market buys.

[5]Technically, while it was realisable on other markets *before* the batch, by the time Feedlot trades that opportunity has already been taken up by CoW solvers themselves.
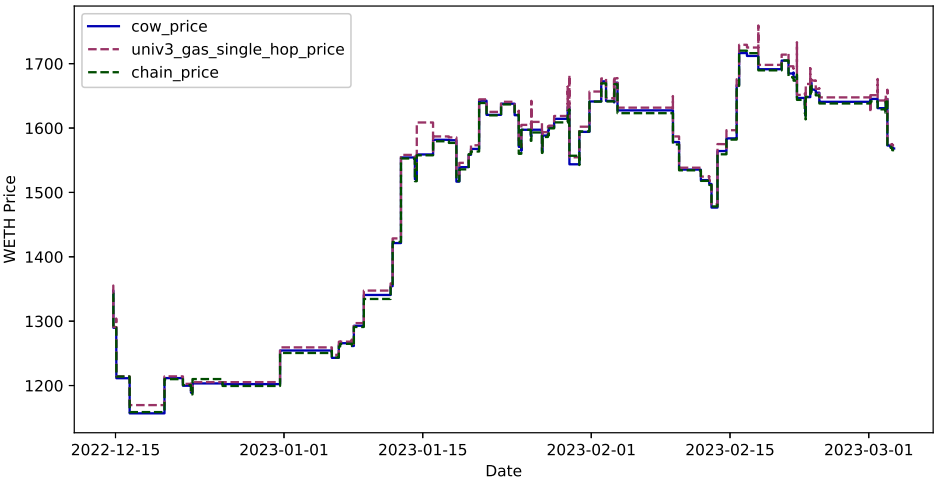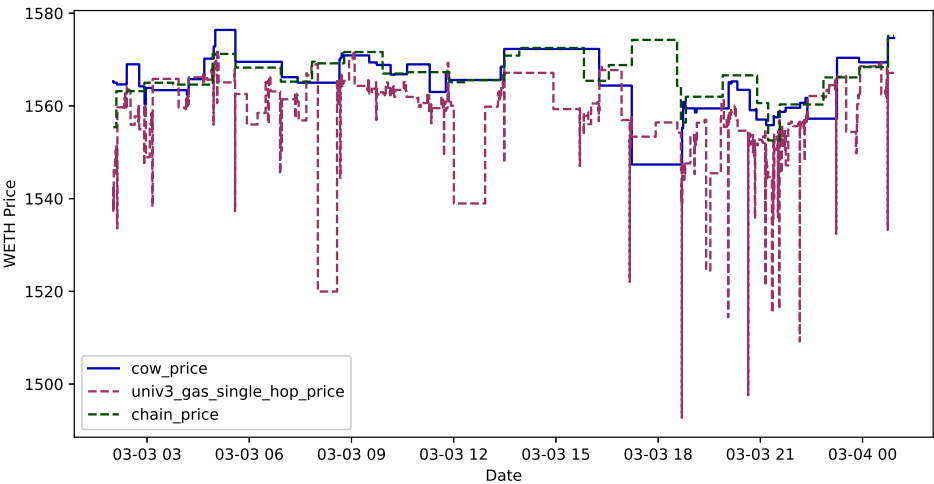
Figure 6: buy side



Figure 7: sell side

**Dataset.** Our dataset consists of Ethereum log data, which records certain on-chain events such as trades. It was sourced from The Graph, a blockchain data indexing service.[6] More precisely, we used the Univ3, CoW, and Chainlink subgraphs curated by Messari, an analytics provider.[7] Full details and our codebase are available [WHERE?].

## 7 Conclusion

A safe implementation of a Feedlot AMM on the subscription model would provide traders with cheap trading at up-to-date instantaneous market prices. Unlike traditional static CFMMs, Feedlot protects LPs from adverse selection, while still offering an option to enjoy flexibly priced portfolio management. It would also yield a new revenue stream for CoW protocol in the form of oracle fees.

The question of whether a safe implementation is possible requires further research. While the guarantees of CoW itself provide that a Feedlot oracle is always 'truthful,' the difficult problem of preventing or discouraging the reporting of an 'artificial' (albeit truthful) price — a price under a centralised influence by a manipulatin attacker — remains a challenge. A team wishing to implement a Feedlot AMM must take great care to ensure that it does not make the CoW UCP an economically viable target for manipulation, inviting harm on Feedlot LPs and users and stakeholders of CoW protocol alike. Simply put, no one would want to provide liquidity to a pool which is forced to accept trades at any price quoted to it.

In §4.2, we have established that some kind of volume controls are a necessary condition for manipulation resistance. However, naïvely limiting volume in terms of the volume on CoW is not sufficient, since this quantity is itself prone to relatively low-cost manipulation. Furthermore, if the scale of traditional derivative markets is anything to go by, there is likely to be interest in increasing the allowable volume to many times the volume of the CoW batch itself. We recommend that further resources be committed to research mitigation and enforcement methodology, with some more specific suggestions outlined in §4.3.

## References

[1] Hayden Adams, Noah Zinsmeister, and Dan Robinson. *Uniswap v2 core*. 2020. URL: https://uniswap.org/whitepaper.pdf.

[2] Bruno Biais, Larry Glosten, and Chester Spatt. "Market microstructure: A survey of microfoundations, empirical results, and policy implications". In: *Journal of Financial Markets* 8.2 (2005), pp. 217–264.

[3] Hamda Al-Breiki et al. "Trustworthy Blockchain Oracles: Review, Comparison, and Open Research Challenges". In: *IEEE Access* 8 (2020), pp. 85675–85685. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2992698.

[4] CryptoCat. *https://cryptocatvc.medium.com/how-to-lose-2-5m-of-your-users-funds-37e02cdb08ec*. 2020. URL: https://cryptocatvc.medium.com/how-to-lose-2-5m-of-your-users-funds-37e02cdb08ec.

[5] Philip Daian et al. "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 910–927. arXiv: 1904.05234 [cs.CR].

[6] Michael Egorov. "StableSwap-efficient mechanism for Stablecoin liquidity". In: (2019). URL: https://berkeley-defi.github.io/assets/material/StableSwap.pdf.

[7] Shayan Eskandari et al. "SoK: oracles from the ground truth to market manipulation". In: *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*. 2021, pp. 127–141. arXiv: 2106.00667 [cs.CR].

[8] Matheus V. X. Ferreira and David C. Parkes. *Credible Decentralized Exchange Design via Verifiable Sequencing Rules*. 2022. arXiv: 2209.15569 [cs.GT].

[9] fleupold. *CoW Native AMMs (aka Surplus Capturing AMMs with single price clearing)*. 2022. URL: https://forum.cow.fi/t/cow-native-amms-aka-surplus-capturing-amms-with-single-price-clearing/1219.

[10] Sean Foley, Peter O'Neill, and Tālis Putniņš. *Can markets be fully automated? Evidence from an "Automated market maker."* 2023. URL: https://github.com/petero1111/website/raw/gh-pages/AMM_Paper_Foley_ONeill_Putnins_2023.pdf.

[11] Mark B Garman. "Market microstructure". In: *Journal of financial Economics* 3.3 (1976), pp. 257–275.

[12] Jonathan Heiss, Jacob Eberhardt, and Stefan Tai. "From Oracles to Trustworthy Data On-Chaining Systems". In: Atlanta, GA, USA. Atlanta, GA, USA: IEEE, 2019, pp. 496–503. ISBN: 978-1-7281-4694-2. DOI: 10.1109/Blockchain.2019.00075.

[13] Max Holloway. *The Value of Nontoxic Orderflow to the Uniswap Protocol*. 2023. URL: The%20Value%20of%20Nontoxic%20Orderflow%20to%20the%20Uniswap%20Protocol.

---

[6] https://thegraph.com/
[7] https://thegraph.com/explorer/subgraphs/ELUcwgpm14LKPLrBRuVvPvNKHQ9HvwmtKgKSH6123cr7

[14]   josojo. *MEV capturing AMM (McAMM)*. 2022. URL: https://ethresear.ch/t/mev-capturing-amm-mcamm/13336.

[15]   Albert S Kyle and Sathish Viswanathan. "How to define illegal price manipulation". In: *American Economic Review* 98.2 (2008), pp. 274–279.

[16]   Bowen Liu, Pawel Szalachowski, and Jianying Zhou. "A first look into defi oracles". In: *2021 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. IEEE. 2021, pp. 39–48. arXiv: 2005.04377 [cs.CR].

[17]   Stefan Loesch et al. *Impermanent loss in uniswap v3*. 2021. arXiv: 2111.09192 [q-fin.TR].

[18]   Jason Milionis et al. *Automated market making and loss-versus-rebalancing*. 2022. arXiv: 2208.06046 [q-fin.MF].

[19]   Roman Mühlberger et al. "Foundational oracle patterns: Connecting blockchain to the off-chain world". In: *Business Process Management: Blockchain and Robotic Process Automation Forum: BPM 2020 Blockchain and RPA Forum, Seville, Spain, September 13–18, 2020, Proceedings 18*. Springer. 2020, pp. 35–51.

[20]   nikete. *MEV Minimizing AMM (MinMEV AMM)*. 2022. URL: https://ethresear.ch/t/mev-minimizing-amm-minmev-amm/13775.

[21]   Wendy Collins Perdue. "Manipulation of futures markets: redefining the offense". In: *Fordham L. Rev.* 56 (1987), p. 345.

[22]   Alexander Port and Neelesh Tiruviluamala. *Mixing constant sum and constant product market makers*. 2022. arXiv: 2203.12123 [q-fin.TR].

[23]   Venus Protocol. *Venus Protocol Official Statement regarding LUNA*. 2022. URL: https://blog.venus.io/venus-protocol-official-statement-regarding-luna-6eb45c3cb058?gi=1d135334a35a.

[24]   samczsun. *So you want to use a price oracle*. Nov. 2020. URL: https://samczsun.com/so-you-want-to-use-a-price-oracle/.

[25]   Anirudh Suresh, Maher Latif, and Nihar Shah. *So you still want to use a price oracle*. Aug. 2022. URL: https://jumpcrypto.com/so-you-still-want-to-use-a-price-oracle/.

[26]   *Synthetix Litepaper*. 2023. URL: https://docs.synthetix.io/synthetix-protocol/the-synthetix-protocol/synthetix-litepaper.

[27]   DODO Team. *DODO — A Next-Generation On-Chain Liquidity Provider Powered by Proactive Market Maker Algorithm*. URL: https://dodoex.github.io/docs/docs/whitepaper/.

[28]   vbuterin. *Let's run on-chain decentralized exchanges the way we run prediction markets*. 2016. URL: https://www.reddit.com/r/ethereum/comments/55m04x/lets_run_onchain_decentralized_exchanges_the_way/.

[29]   Anthony Lee Zhang. "Competition and manipulation in derivative contract markets". In: *Journal of Financial Economics* 144.2 (2022), pp. 396–413.

[30]   Liyi Zhou et al. "High-frequency trading on decentralized on-chain exchanges". In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 428–445. arXiv: 2009.14021 [cs.CR].