On-chain Perpetual Futures

Audaces Foundation

April 2021

Abstract

This paper gives an overview of the implementation as well as the economics behind the upcoming Audaces Perpetual Futures protocol. The design implements a Virtual Automated Market Maker (vAMM), drawing inspiration from Ethereum's Perpetual Protocol and leveraging the Solana blockchain's state-of-the-art performance. The Audaces Perpetual Futures automated market maker contributes a novel balanced funding protocol which aims to protect the market's insurance fund against sudden trend shifts, while enabling dynamic market liquidity scaling on a vAMM for the first time.

Contents

1 Introduction

This paper introduces a new perpetual protocol built on the Solana blockchain. It aims to provide a high level overview of how the protocol is implemented and the innovations it presents compared to existing implementations of perpetual futures.

2 Virtual Automated Market Maker

The concept of a Virtual Automated Market Maker (vAMM) was first introduced by Perpetual Protocol [3]. A vAMM works similarly to a traditional AMM with a constant product curve (e.g a $x * y = k$) [5], except that as the name suggest, the liquidity in the AMM is virtual. The liquidity is set at the creation of the market and all trades happen in the vAMM. In practice, the state of each market i.e the vAMM is stored in the data of a Solana account [1].

One interesting property of a vAMM is that that any metric can become a future as long as an oracle exists for it. For instance, in a `BTC/USDC` futures market, no actual `BTC` is exchanged and all transactions are conducted in `USDC`. The market uses its own price discovery mechanism (the vAMM) to perform conversions between virtual assets and the quote asset.

A key variable which determines the vAMM's price inertia is k. In the conventional constant product curve model, k is an immutable variable throughout the lifetime of the market. The choice of value for k is a compromise between lower slippage with higher values, and ease of arbitrage at lower values. In essence, k can be thought of as a measure of the market's available *virtual liquidity*. Moreover, higher values of k decrease the efficiency of the rebalancing mechanism described in Section 5. In general, higher values of k require more trading volume to properly function. Section 6 looks at a solution to the problem of k's immutability which is deeply related to the rebalancing mechanism.

3 Liquidation

As any protocol that brings leverage, the liquidation risk is the highest risk of the protocol. This is why the entire protocol was designed to optimize the liquidation speed.

In order to efficiently liquidate positions, the protocol proposes an alternative to the naive design which iterates over all known accounts in order to find candidates for liquidation, and then individually liquidates all those positions in almost as many transactions. In order to fully leverage the performance of the Solana blockchain, the ideal goal should be to perform liquidation operations across the whole market in one instruction.

However, an obstacle to this is that Solana enforces a limit to the number of computation cycles that can be consumed by a single instruction. This means that any approach which iterates over all open positions is bound to fail at some point. To solve this problem, the Audaces perpetual futures market makes use of two central open positions indexes (one for short positions and one for longs) which are called the *Liquidation Trees* (LTs).

When opening an order, it is possible to precompute the price threshold at which liquidation should occur. This liquidation index is then inserted into a *critbit* tree, which is stored in a paged memory heap on chain. The identifier of the created leaf is then recorded into the user's account. When a user interacts with this position in the future, the smart contract will look for the corresponding leaf. If it cannot find it, then it means that the position was liquidated.

The LT itself has inner nodes which track the total amount of collateral and virtual assets that its descendant open positions hold. Since the critbit tree is naturally ordered by liquidation index, the liquidation operation becomes very efficient. The smart contract walks down the liquidation tree and then only has one edge to cut off to remove an arbitrary number of open positions. Since the liquidation index is stored as a 64 bit integer, the tree itself can only be at most 64 layers deep, which means that the tree walking operation is bounded and liquidations can be proven to succeed in any context by testing this worst-case scenario.

A limitation to this model is that once liquidation happens it becomes impossible to keep track of when a particular order was liquidated. It is easy to prove why a particular order was liquidated, but the absence of this information means that it becomes impractical to refund the maintenance margin to liquidated users. However, the market's liquidation performance can alleviate this performance by allowing for lower maintenance margins with manageable risk for the insurance fund.

In practice, fast liquidation cranking servers will issue liquidation transactions at every time slot, aiming for every new Solana block to contain at least one liquidation transaction.

The *Liquidation Tree* architecture effectively implements the fastest possible liquidation solution on Solana as the bottlenecks for faster performance become the oracle's update frequency and Solana's slot validation time. This is why Pyth Network was chosen as the underlying oracle solution since it promises time resolutions on par with slot times.

Therefore, the only limiting factor to the speed of liquidation is the speed of the oracle meaning that if the oracle updates every x ms and the Solana block time is y (at the time of writing $y \sim 400ms$) the frequency of liquidations will

be $\frac{1}{max(x,y)}$. This means the liquidation system has no issue scaling, as unlike other margin trading protocols on Solana the liquidators don't have to iterate over all accounts.

The liquidation is executed relatively to the index (oracle) price. This, in contrary to a liquidation relative to the market price, alleviates the possible problem of a liquidation snow-ball effect where each liquidation of a Short (for example) only drives the market price higher up, provoking new liquidations, and so forth and so on.

Finally, as the LTs hold information about the current open positions, the 10Mb Solana account memory limit could eventually be reached as the market grows. In order to circumvent this limitation, a memory paging mechanism is implemented that splits the memory slots that hold the LT information over multiple accounts. Nevertheless, using tens of accounts as input for a Solana instruction could still result in unwanted behaviour, the computation cycles limit alone could already hinder the deserialization of the memory.
For this reason, we make use of so-called LT instances. For one market there can be multiple instances that each store the 2 LTs (one for Shorts, one for Longs), each possibly making use of multiple memory pages (e.g. accounts). This makes it possible for each instance to use less than 10 Solana accounts while the complete open positions information for the market spans over tens or hundreds of memory pages. The only limitation in scale is the frequency at which the liquidation can be cranked as it is executed on an instance at a time, in other words, on a pair of LTs at a time.

4 Funding

In order to be able to make liberal use of efficient data structures, the Audaces Perpetuals market doesn't extract funding directly from a user's open positions. Instead, funding is paid from a user account's balance, leaving the positions unaffected. This means that users are required to maintain enough balance to pay for potential funding in order to avoid having their positions liquidated. A consistently well arbitraged market will require less funds locked in user accounts, minimizing the associated impermanent loss.

In practice, funding happens in three stages, and is based on the formula shown in Figure 1. The first operation samples the market's funding ratio at short but regular intervals in order to get an accurate estimate of the time-weighted moving average of the relative difference between market price and index price. The second operation happens at much longer interval and calculates the final value of the funding ratio for the past round. At this stage, the imbalance between long and shorts is recorded and is used to cap the funding that is received by users to the funding that is paid. This is to ensure that

$$f(t_0 + P) = \frac{P}{24} \sum_{i=0}^{N} \frac{1}{N} \frac{m - o}{o} \left(t_0 + \frac{iP}{N} \right) \tag{1}$$

Figure 1: The funding ratio f, with P the funding period (in hours), N the number of samples in a given period, m the market price function, o the index price (or oracle) function

the insurance fund never pays out funding. The final operation happens on a per-user account basis and extracts the relevant funds from the user's account, liquidating their positions if necessary.

5 Rebalancing

The nature of a vAMM means that if every actor chooses to close their positions, the market resets to its origin price. This behaviour indicates that the core design lacks time symmetry, which has important consequences on the stability of the system as a whole. If the price of an asset has increased over time since market launch, this translates into the fact that a perfectly arbitraged market has more longs than shorts. As a consequence, if the current vAMM price is under the oracle price (but still above the origin price), funding would have to be paid from the insurance fund. Since there are more longs than shorts, the protocol would have to inject insurance fund money to pay the longs the funding they deserve. This problem only gets worse and worse over the lifetime of a market, especially as the price diverges from the origin price.

To tackle this, one solution is to pay out funding from open positions and not from the insurance fund. Although this is much safer for the insurance fund, this means that users don't get as much funding as they would expect and the market incentives for arbitrage are at risk of debilitating significantly over time. Rebalancing comes in to work against this imbalance and guarantees, as the market starts to accumulate rebalancing funds, that the equilibrium price never strays too far from the mark price (the threshold which triggers rebalancing is currently around a 10% difference).

Rebalancing works by setting aside some funds in order for the vAMM to buy its own funding-exempt fully collateralized positions. In order to direct the equilibrium price towards the index price, the vAMM will automatically manage its own implicit positions. This means that these positions will never get liquidated, which is why there is no need to keep track of them explicitly. If there are too many longs, the vAMM will buy out longs (or close its own shorts) when users attempt to close their longs, or attempt to open shorts, a bit like a bot in an order-book based market. In practice this means that some operations don't change the price as much as they should, which offers the possibility to arbitrageurs to profit from this, creating an additional incentive to rebalance the market when rebalancing funds are available.

$$
\begin{aligned}
marketPrice &= \frac{vQuote}{vBase} = \frac{k}{vBase^2} \\
bias &= \frac{currentPrice}{eqPrice} = \left(\frac{vBaseEq}{vBase}\right)^2 \\
vBase &= vBaseEq - longs + shorts = vBaseEq - \Delta \\
bias &= \left(\frac{vBase + \Delta}{vBase}\right)^2 \\
bias &= \left(1 + \frac{\Delta}{vBase}\right)^2
\end{aligned}
\tag{2}
$$

Equation 2 gives the derivation of the core balancing metric, *bias* as the ratio of the current mark price and the so-called equilibrium price. The rebalancing mechanism takes off significant long-term risk from the insurance fund by adapting funding to the bias (which means that in our scenario longs would only get a fair proportion of what the market was actually able to extract from the shorts).

6 A solution to the k problem

A core limitation of the traditional vAMM implementation is that is is difficult to dynamically change the value of k in operation. However, increasing the value of k as a market's volume increases can minimize slippage on larger orders while still being easier to arbitrage during the lower volume inception period. The core problem behind increasing the value of k is that it breaks path independence making it possible for users to directly take profit from the insurance fund. This means that changing the value of k can be a very costly procedure which effectively injects funds into the market from the insurance fund.

6.1 Method and derivation

This cost can be evaluated by comparing two scenarios in which all users close their positions and extract their payout from the market. In the first scenario the value of k remains unchanged, in the second scenario the value of k is first multiplied by a scalar. This change of k is achieved without changing the market price by multiplying both the virtual coin and virtual quote amounts by a scalar p. This means that k is multiplied by p^2.

Let x and x_0 be the market's current and equilibrium amounts of virtual coin, and y and y_0 the related amounts of quote currency:

$$
\begin{cases}
xy = x_0 y_0 \\
x = x_0 - \Delta \\
\Delta = longs - shorts
\end{cases}
\tag{3}
$$

By leaving the value of k unchanged, closing all positions amounts to adding back Δ to x which reverts the vAMM back to x_0 and y_0. This reversal can happen in multiple different ways: the order in which each position is closed matters in determining each position's associated payout. For instance, it is more profitable for a user holding a long position to wait for someone to close their short position before closing their own. However, a core property of any vAMM is *path independence*. This means that the total payout of all users doesn't depend on the sequence of operations: only the end state matters. This allows for a reduction in the analysis to a trivial path: all longs are closed, and then all shorts are closed. Path independence even allows for a consideration of any vAMM state as equivalent to a market containing only two open positions: one short and one long. Therefore the total payout δ that users can extract from the vAMM is the sum of the long and short payouts in this simplified scenario.

We begin by mutliplying the vAMM state by p, which yields a state of (px, py). This has the effect of changing k to p^2k with the market price remaining constant. Then, we close the long position of size l and solve for the resulting (x', y') state:

$$\begin{cases} x'y' = p^2xy \\ x' = px + l \end{cases} \tag{4}$$

$$\implies y' = \frac{p^2xy}{px + l} \tag{5}$$

We use K_l to refer to the opening virtual quote amount for the long position, and K_s for the short position. The associated payout P_l for the long position is then given by:

$$P_l = py - y' - K_l \tag{6}$$

$$= \frac{pl}{px + l}y - K_l \tag{7}$$

We then close the short position of size s and solve for the resulting (x'', y'') state:

$$\begin{cases} x''y'' = x'y' \\ x'' = x' - s \end{cases} \tag{8}$$

$$\begin{cases} x'' = px + \Delta \\ y'' = \frac{p^2xy}{px+\Delta} \end{cases} \tag{9}$$

The associated payout P_s for the short position is then given by:

$$P_s = K_s - (y'' - y') \tag{10}$$

$$= K_s - p^2xy\left(\frac{1}{px + \Delta} - \frac{1}{px + l}\right) \tag{11}$$

Then, the total amount of funds which are extracted from the vAMM is given by $\delta(p, \Delta)$:

$$\delta(p, \Delta) = P_s + P_l \tag{12}$$

$$= K_s - K_l + \frac{p\Delta}{px + \Delta} y \tag{13}$$

To then evaluate the direct cost of changing k to pk one can just look at $C(p, \Delta) = \delta(p, \Delta) - \delta(1, \Delta)$:

$$C(p, \Delta) = \left(\frac{p}{px + \Delta} - \frac{1}{x + \Delta} \right) \Delta y \tag{14}$$

One can then solve for $C(p, \Delta) = 0$ which is equivalent to $\Delta^2 = 0$ which means that $\Delta = 0$ is the unique solution to this problem in the current framework. This double solution hints at the fact that allowing ourselves to change the amplitude of the positions (effectively introducing a Δ') gives a more general solution, but the problem with non-Δ-zero solutions is that the cost burden of the k transform becomes asymmetrical for the user, favoring one side over the other.

Fortunately, attaining a state in which $\Delta = 0$ is possible by leveraging the rebalancing mechanism discussed in section 5. This fact also offers insight into why rebalancing is so important: in balanced conditions, a vAMM acts like an orderbook-based market in that actors trade directly against each other instead of against the vAMM itself.

6.2 Notes on payouts after k transforms

Using the algorithm described in subsection 6.1 strictly alters the payouts given a particular vAMM path. In practice, this means that the same sequence of operations with a higher value of k diminishes the associated amplitude of profits and losses for different users, while overall balancing out. Intuitively, this is due to the fact that a higher value of k means that the vAMM's state will span a smaller interval of market prices given the exact same operations, due to lower slippage.

However, if the value of k is chosen to reflect the market's capacity for arbitrage, market incentives will in practice always compensate for this difference by providing a market price which still follows the index price. This means that, in practice, positions will actually tend to pay out *more* due to lower slippage on market operations. This is why breaking strict payout invariance across k transforms is considered a non-issue in this case. What is really happening is that the market itself becomes more exposed to *arbitrage failure*, a risk which is described in greater detail in subsection 10.1.

7 Insurance Fund

The protocol is backed by an insurance fund. The insurance fund will receive a fraction of all the fees from the protocol. To see how the insurance fund behaves in different scenarios please refer to **section 9**.

8 Cranking

Certain functions are required to be run on schedule to maintain the protocol. Their execution is referred as "cranking". Cranking is required for: liquidations, funding rates and garbage collection. Note that these instructions are permissionless, it means that any Solana wallet can "crank" them. Cranking requires to pay gas fees for the transactions but crankers are rewarded in USDC for the transactions they crank.

The process that need to be cranked are (in no particular order):

- Liquidations
- Funding rates
- Garbage collection

A Rust executable as well as a Typescript client that automate the cranking can be found on the protocol's repository.

9 Quantitative simulations

The following figures show the evolution of the market state during simulation runs that aim at reproducing close-to-real market conditions. This testing environment is built upon the solana-program-test library [4] which instantiates a local Solana network and loads the perpetuals smart contract to it. By interacting with this system the quantitative behaviour of the vAMM and the different rebalancing mechanisms can be simulated, and also fuzz and test the program for potential errors.

The plotted data has been scaled in order to emphasize the visual correlation of events in the market, the absolute values are therefore irrelevant.

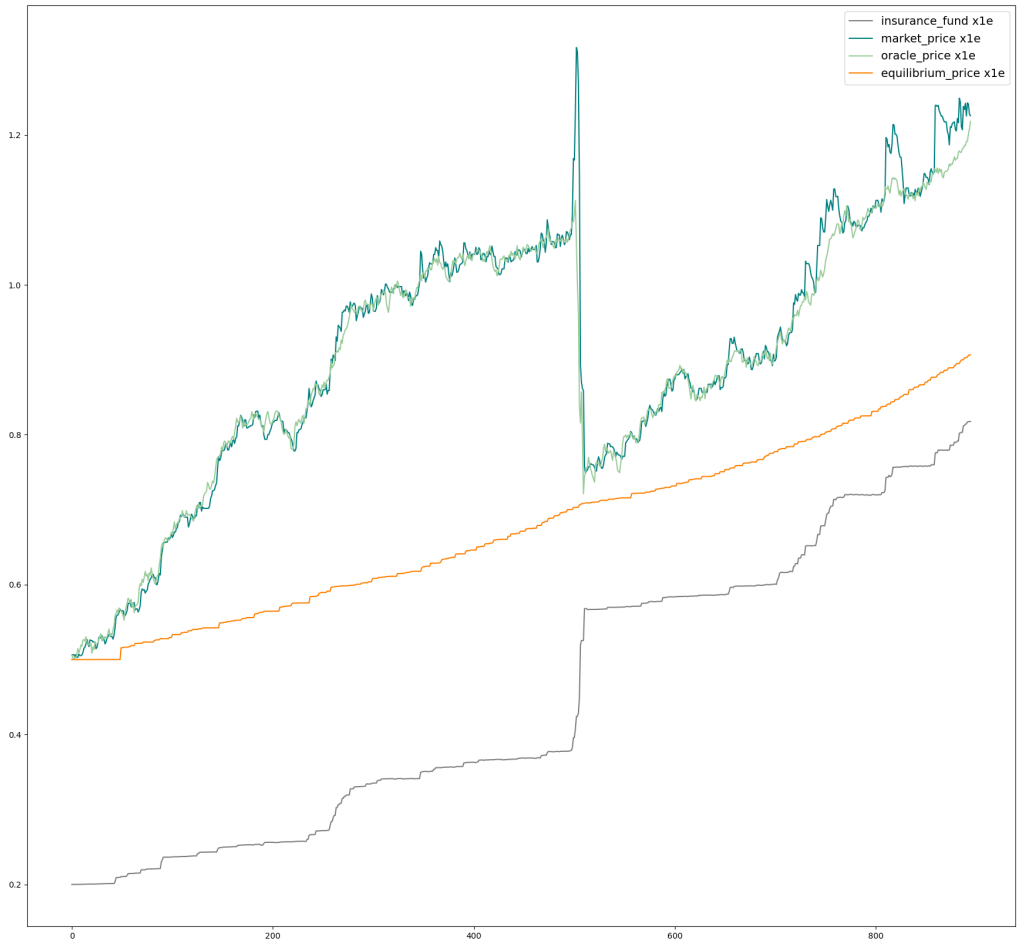

Figure 2: The evolution of the insurance fund (grey), market price (blue), index price (green) and equilibrium price (orange) over time (e.g. instructions) in a monotone market that is interrupted by a sudden price drop.

In this scenario, the abrupt simulated crash resulted in a spike in liquidations. This is also illustrated by the drop of the total open longs amount. Moreover, it can be noted in the figure below that during the approximately 30 first instructions, the equilibrium price remains constant while the rebalancing is growing. This is due to the tolerance margin in which the equilibrium price finds itself relatively to the market price - when the market price escapes from that margin, the rebalancing gets activated.

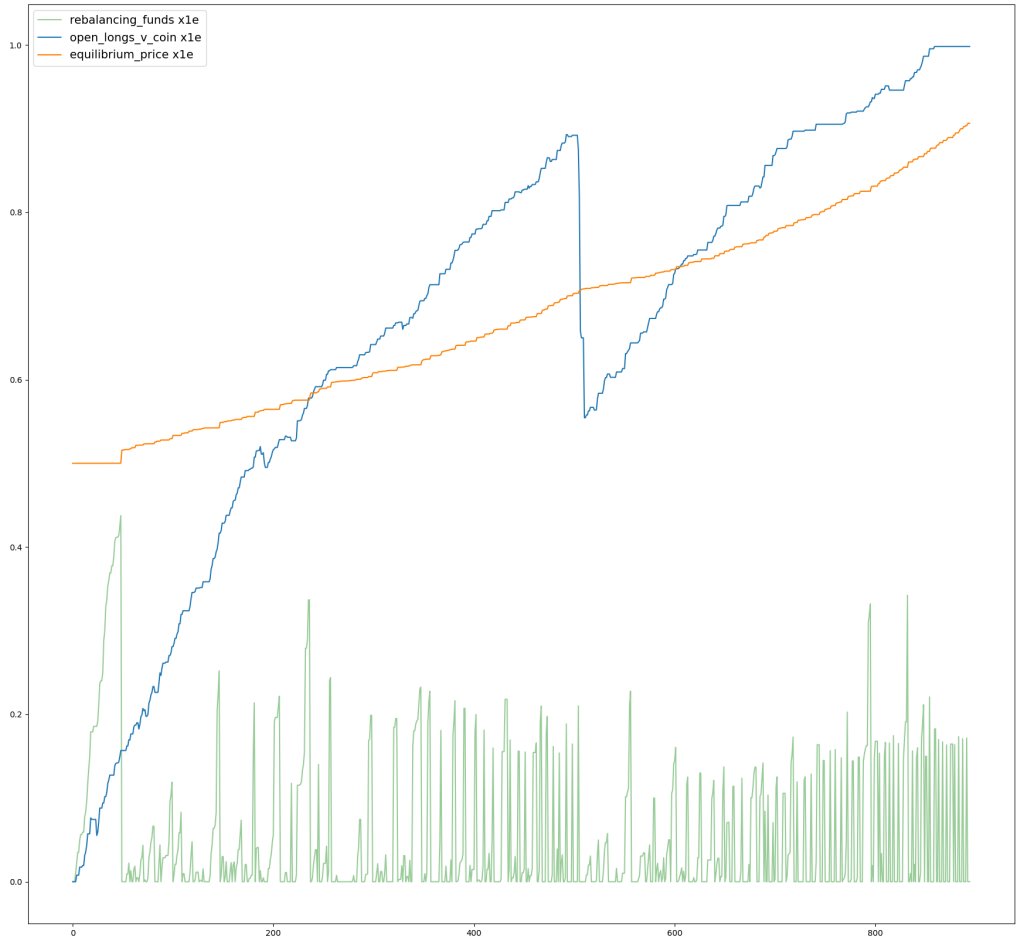

Figure 3: The evolution of the rebalancing fund (green), total virtual base amount invested in open positions (blue) and the equilibrium price (orange) over time (e.g. instructions) during the same simulation run as in the figure 2. The semi-regular drops in the rebalancing funds reflect the rebalancing mechanism that pushes the equilibrium price towards the market price.

10 Risk and Limitations

Using the Audaces Perpetuals market is not without risk. Care has been taken to minimize the impact of the following sources of risk on the user, strictly improving the existing state-of-the-art decentralized perpetuals markets. There are two broad categories of risk involved in our analysis: intrinsic risk and platform risk.

10.1 Intrinsic risk

Insufficient volume for rebalancing Although care will be taken to tweak the market's parameters in order to stimulate rebalancing, rebalancing performance is intrinsically linked to volume. To be exact, rebalancing performance is proportional to the ratio between price change and volume over a given period of time. If the market becomes sufficiently imbalanced, funding performance can greatly decrease, which means that arbitrage incentives become lopsided between longs and shorts.

Arbitrage failure If for any reason the market price begins to completely diverge from the index price, a situation might arise in which positions with locally negative payouts cannot be closed or liquidated. This is due to the fact that payout depends on the market price whereas liquidation depends on the index price. In essence, this means that users will have to wait for the market price to converge again before closing their positions. This risk increases when the market is sufficiently imbalanced: if every user attempts to close their positions, the market price will not converge to the index price and some users might be able to extract their payout whereas others might be stuck until market arbitrage improves again. This is why index-based liquidation requires rebalancing.

10.2 Platform risk

Oracle failure If for any reason the oracle ceases to function, or its confidence metric begins to drastically decrease over a long enough period of time, the liquidation mechanism becomes entirely crippled. Once the oracle comes back on line, the liquidation engine will directly tap into the insurance fund to buffer any potential loss of funds.

Solana network failure If for any reason the network state becomes corrupted, or the network itself becomes unresponsive with too many transactions getting dropped, the liquidation engine can be slowed down which locally incurs extra risk on the insurance fund.

Nefarious leader If the current leader validator decides to drop all transactions associated with liquidations, the liquidation engine can be effectively disabled for an epoch which incurs extra risk on the insurance fund. According

to the official Solana documentation [2], an epoch is about 100 slots long, which means that a single leader can disable liquidation for about a minute. A nefarious leader validator can also censor the opening of certain positions, or simply reorder transactions in such a way that it enables an artificial market price bias to be constructed, a bias which can then be used by the validator to effectively front-run other orders. In order to mitigate this risk, a slippage threshold can be set by users to prevent orders from going through at unexpected entry prices.

11 Conclusion

The protocol described in this white paper solves the main issues encountered by previous implementations of vAMMs and sets a framework for future protocols to build upon. The main innovations of the protocol dwell in the dynamic equilibrium price that mitigates the imbalance between longs and shorts. Because the imbalance is mitigated, the protocol can change k without interruption or breaking path independence. This means that the protocol can adapt its liquidity as volume grows. The protocol also implements a capped funding rate to protect the insurance fund, however, as the *long/short* imbalance is mitigated the funding rate is likely to be similar to a CEX funding rate. The liquidation engine was designed for speed and scalability and the liquidation tree architecture implements the fastest possible liquidation solution on Solana and DeFi in general. These two safeguards guarantee that the insurance fund is never at risk even during volatile times where liquidation cascades can be triggered and the mark price can deviate significantly from the index.

Audaces foundation has decided to not release a token and join forces with early developers on Solana. Bonfida being one of them, Audaces foundation decided to pledge all the revenues to the FIDA buy and burn smart contract. As a result FIDA holders will benefit from reduced fees when trading on Audaces protocol. While the Audaces protocol is immutable, the smart contract that receives the fees might be modified via a governance.

References

[1] *Accounts: Solana Docs*. URL: https://docs.solana.com/developing/programming-model/accounts.

[2] *Leader Rotation: Solana Docs*. URL: https://docs.solana.com/cluster/leader-rotation.

[3] Perpetual Protocol. *Litepaper*. URL: https://docs.perp.fi/library/litepaper.

[4] *Solana Program Test Library*. URL: https://docs.rs/solana-program-test/1.6.9/solana_program_test/.

[5] Yongge Wang. "Automated Market Makers for Decentralized Finance (DeFi)". In: (). URL: https://arxiv.org/pdf/2009.01676.pdf.