Medallion: A competition platform to reduce LVR

mxwtnb[1] and cryptoma20[2] and easyrider[3]

mxwtnb@charm.fi[1], tom@charm.fi[2], dz@charm.fi[3]

September 7, 2024

Abstract: Medallion is a DEX where LVR reduction strategies compete to earn the highest yields for liquidity providers (LPs). It generalizes auction-managed AMMs, so that the widest range of LVR reduction strategies can compete on an open and permisionless platform. The winning strategy calculates a different fee for each swap in a liquidity pool, pays the highest rent to LPs, and earns all the swap fees in the pool. This paper explains how Medallion increases LP yields, and describes some of the LVR reduction strategies that can compete on the platform.

1 Introduction

Loss-versus-rebalancing (LVR) is a significant source of losses for LPs, far exceeding the losses from impermanent loss, sandwich attacks, and front-running attacks [5]. Many strategies are being researched to reduce LVR, such as dynamic fees, auctions, and using an oracle; but given the size and complexity of LVR, it's unlikely one strategy will always be the best. A mechanism is required to automatically select the best strategy from all the possible strategies, in order to recover the largest amount of LP losses.

The am-AMM [1] is one such mechanism. It supports some strategies but lacks support for many others. Medallion lets users create their own *strategy contracts* to customise the am-AMM, in order to support more strategies, and increase the design space for new strategies to emerge. This results in more competition, and higher yields for LPs.

2 Prior work

LVR are losses incurred by AMM LPs due to stale prices that are picked off by better informed arbitragers [8]. Most of the arbitragers' profits are extracted by block builders [10], and LVR reduction strategies redistribute profits from block builders to LPs. *Dynamic fees* and *application layer auctions* are the most common categories of LVR reduction strategies.

Dynamic fees detect signals for toxic order flows, and charge higher fees for swaps that are more likely to be toxic. *Dynamic fees* can use auto-correlation signals [9] to charge higher fees if a swap is in the same direction as the previous swap; volatility signals to charge higher fees when the volatility is high [2]; router signals to charge lower fees for swaps originating from an address that routes retail trades [3]; and machine learning signals to charge different fees depending on the patterns detected from machine learning algorithms.

In *application layer auctions*, users bid for the opportunity to execute swaps, set fees, and capture MEV (Maximal Extractable Value). The applications pay the bids to LPs as extra yields. Examples of *application layer auctions* include McAMM [7], where bidders bid for the right to execute the first swap in a block; FM-AMM [4], where bidders bid for the right to execute all the swaps in a batch at the same price; Sorella [12], where bidders bid to execute the fist swap and a batch of swaps; and the am-AMM [1], where bidders bid to set the same fee for all the swaps within a block.

Less common approaches include oracle aware AMM [8] and Hybrid AMMs [6].

Medallion combines the best features of *application layer auctions* and *dynamic fees*.

3 Medallion

Medallion is a DEX where LVR-reduction strategies compete to earn the highest yield for LPs. It has a public auction where competing strategists submit bids - the highest bidder pay their bid to LPs, attach their strategy to a pool, and earn all the pool's swap fees.

Medallion's approach is similar to the am-AMM, with one significant difference. In Medallion, anyone can write a *strategy contract* to reduce LVR, and bid to attach the contract to a pool. The contract implements a `calculateSwapFee()` function to calculate the fee for each swap. The fee can be different, depending on the size, direction and sender of the incoming swap; and can change automatically without requiring a separate transaction.

Strategy contracts customise the am-AMM, similar to hooks customising a Uniswap pool.

4 Strategy Contract

Figure 1: The strategy contract

Strategy Contracts are user deployed contracts that execute pre-defined logic to calculate a different fee for each swap in a liquidity pool. Here are some examples of the LVR reduction strategies that can be built using *Strategy Contracts*:

- Dynamic fee strategies:

  A manager noticed a swap in the same direction as the previous swap is more likely to have higher LVR. They write a *Strategy Contract* to set a swap fee of 1%, and 3% if the next swap is in the same direction.

- Auction strategies:

  A manager believes the first swap in a block is more likely to be an arbitrage trade, and build an auction where arbitragers bid for this swap. The winning bidder's address will be whitelisted by a *Strategy Contract* to execute this swap, and all transactions submitted before this swap will revert.

- Signalling strategies:

  A manager is looking for LVR signals, and found 0x1234 is an arbitrage bot generating toxic flow, and 0x5678 is a router routing retail trades. They write a *Strategy Contract* to charge a 5% swap fee for 0x1234, a 0.3% fee for 0x5678, and a 1% fee for everyone else.

- Oracle strategies:

  A manager writes a *Strategy Contract* to charge 3% fee for swaps that move the AMM price towards a Binance spot price, and 1% fee for other swaps.

- Volatility strategies:

  A manager believes the market will be volatile over the next two days, and estimates swappers are willing to pay 50% more fees during this period. They write a *Strategy Contract* to set the fee to 3% for the next two days, and 2% afterwards.

- Arbitrage strategies:

  A manager writes a *Strategy Contract* to charge 3% fee if the DEX price is between 10 and 20. If the price moves to 10, the manager can do zero-fee swaps to capture all the arbitrage profits when the CEX price moves between 10 and 10.3.

- MEV tax strategies:

  For chains that support competitive priority ordering [11], a manager can create a *Strategy Contract* to read the chain's priority fee. This allows a high MEV tax to be charged for the first swap in a block to reduce LVR, and a low MEV tax for subsequent swaps to incentivise uninformed flows.

- Replicating other AMMs:

  The manager can replicate an AMM with a static fee (eg the am-AMM) by writing a *Strategy Contract* that sets the same fee for all swaps within a block.

- Replicating liquidity distributions:

  Assume ETH price is 2000, and a manager writes a *strategy contract* to replicate an order book distribution of 1 ETH bid at 1980 and 1 ETH bid at 1960. The first 1 ETH will sell at a 1% swap fee; the next 1 to 2 ETH at 2%; and swaps selling any remaining ETH will be rejected.

- Combination strategies:

  A manager building a dynamic fees strategy can supplement their earnings by auctioning off the rights to execute a swap.

In the future, it is likely Medallion users will create a wide range of independently researched *strategy contracts*.

5 Public Auction

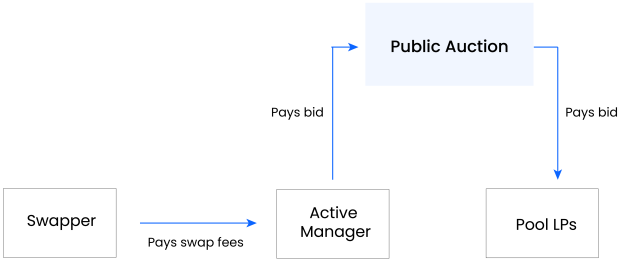

Figure 2: The public auction

Anyone can deploy their own *strategy contract*, and use Medallion's public auction to bid for the right to attach their contract to a pool. The highest bidder becomes the *active manager* and pays rent (i.e. the amount bid) to active LPs at each block, and anyone can place a higher bid at any time to become the new *active manager*. The *active manager*'s contract automatically sets a different fee for each swap, and the *active manager* earns all the swap fees generated by the pool. LPs receive the *active manager*'s rent as extra yields.

Here is an example of how the auctions works:

1. Manager A created a strategy contract that generates expected fees of $150 a day, and placed a bid of $100 per day to become the active manager. $100 was paid to LPs as rent, and the manager earns a profit of 150 - 100 = $50 a day.
2. Manager B built a similar contract and is willing to make less profit. They bid $110 a day to become the new active manager. $110 was paid to LPs, and the manager earns $40 profit.
3. Manager C creates a better contract that generates expected fees of $300 a day. They bid $200 a day to become the new active manager. $200 was paid to LPs, and the manager earns $100 profit.

6 Implementation

All the functionalities will be implemented in a single contract, the *Medallion hook*. It's a singleton contract that can be attached to Uniswap V4 pools as a hook. It contains the public auction, manages the bids, and the payments to managers and LPs. Fig 3 shows how the *Medallion hook* interacts with other users within the Medallion ecosystem. In the ecosystem:

- Pool creators set up a *Medallion pool* by creating a Uniswap v4 pool attached to the *Medallion hook*.
- For each swap, `beforeSwap()` is called to obtain the swap fee in `getFee()`. The swap fee will be routed by the hook to the highest bidding manager.
- Managers call `modifyBid()` to place or modify a bid, and the hook will pay the bid to the pool's LPs as rent. Managers pay bids and withdraw funds by calling `deposit()` and `withdraw()`.
- Providing liquidity to a *Medallion pool* works the same way as providing liquidity to any other Uniswap V4 pool.
- Swaps are atomic, and work the same way as any other Uniswap V4 pool.

The following sections provides further details of the ecosystem.

6.1 Pool creation

Medallion pools are standard Uniswap V4 pools attached to the *Medallion hook*. Anyone can create a *Medallion pool* by calling `initialize()` in Uniswap V4's `PoolManager` contract, with `hookAddress` pointing to the *Medallion hook* so that the pool can access Medallion's functionalities.

Medallion pools can be created for any pair to tokens and tick spacing, and can be done programmatically or using a front-end.

6.2 Swapping

Swaps are atomic, and work the same way as any other Uniswap V4 pool. Frontends, aggregators, and fillers don't have to implement extra logic to swap.

The Medallion hook implements `beforeSwap()` to fetch the swap fee from the active manager's *strategy contract*, and redirect the swap fees to managers instead of LPs.

6.3 Liquidity Provision

Adding or managing liquidity works the same way as any other Uniswap V4 pool - it can be done using periphery contracts or via a liquidity manager. A frontend can be created to help users find a Medallion pool and provide liquidity.

6.4 Manager and Strategy Contracts

Managers are users who build *strategy contracts* to set the fees of a swap, in order to recover LP losses and

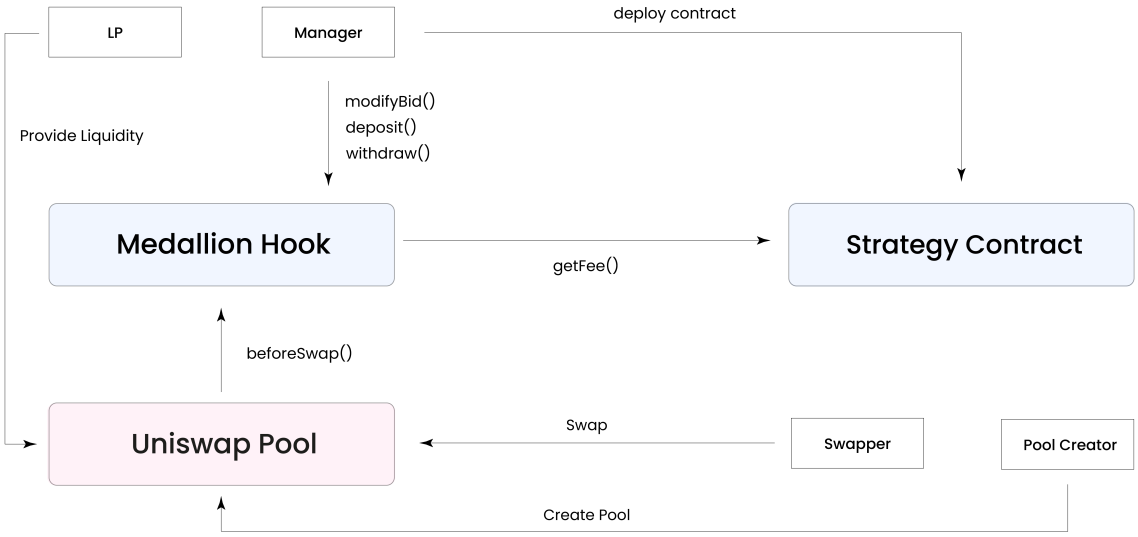

Figure 3: The Medallion ecosystem

reduce LVR. When a user bids to become a manager, they specify the address of a *strategy contract*.

If the user places the highest bid, they become the *active manager*, and their *strategy contracts* will be called on every swap to determine the swap fee. The *active manager* pays the bid to LPs as rent, and earns all the swap fees. They can call `withdraw(amount0, amount1)` to withdraw the fees earned.

strategy contracts must implement the `calculateSwapFee(Z, A, S)`[1] method, because it is called by the *Medallion hook* before each swap to calculate the fee to be charged for the swap. The implementations contain their own logic to determine the fee or to revert swaps.

If a *strategy contracts* doesn't implement the fee properly or sets fees that are too high or too low, LPs will not be affected. It will simply prevent swaps from taking place, and the manager will misses out on revenue. If the manager is blocking swaps within the pool, users can call a `forceSwap()` method in the *Medallion hook* to force a swap with a fixed fee.

6.5 Public Auction and rent payment

Medallion's public auction is implemented as a singleton contract that can be attached to Uniswap V4 pools as a hook. Anyone can submit the highest bid at any time to become the new active manager of a pool. They pay the highest rent per block to LPs.

To bid, users call `MedallionHook.modifyBid(P, S, F, R)` [2]; and existing bidders can modify their bids using `modifyBid(rent, strategyAddress)`. Bidding is permissionless - anyone can submit a bid at any time.

`rent` is the amount a bidder is willing to pay LPs at each block to become the *active manager*, and a bid can be canceled at any time by setting the rent to 0. `strategyAddress` points to a *strategy contract* that implements the fee calculation logic - it is usually the contract built by the bidder.

6.5.1 Rent payment

The manager uses `deposit(amount0, amount1)` to deposit tokens to the *Medallion Hook* to pay rent, and the deposit will be gradually decreased as rent is paid to LPs. If a manager's deposit does not cover the rent, their bid will be ignored, and the next highest-bidding manager who has sufficient deposits will become active. They can call `withdraw(amount0, amount1)` at any time to withdraw their rent.

Rent is paid each block to in-range LPs. The actual transfer of funds is triggered during a relevant transaction, such as before a swap; when the amount of active liquidity changes; or when the manager changes. The total amount of rent owed since the last payment is calculated, and then transferred from the manager's deposits to in-range LPs using the `donate()` method. The amount of rent LPs receive is proportional to the size of their in-range liquidity, and the funds will be transferred using ERC6909 to minimise gas costs.

[1] Z = zeroForOne, A = amountSpecified, S = senderAddress
[2] P = pool, S = strategyAddress, F = feeRecipient, R = rent

7 Conclusion

Medallion increases LP yields by optimising the swap fees generated by a Uniswap V4 pool. Its architecture is decentralised, which means anyone can build any strategy to set the fee of a swap, and then bid in a public auction to recover LP losses and reduce LVR. LPs earn the highest yields from the biggest bids, and strategy builders earn all the fees in a liquidity pool.

References

[1] Austin Adams, Ciamac C. Moallemi, Sara Reynolds, and Dan Robinson. 2024. *am-AMM: An Auction-Managed Automated Market Maker*. Retrieved 2024-08-27 from https://arxiv.org/abs/2403.03367

[2] Ambient. 2023. *Volatility strategy*. Retrieved 2024-09-05 from https://docs.ambient.finance/users/dynamic-fees

[3] Balancer. 2023. *Fee Discount for CowSwap Solvers*. Retrieved 2024-09-05 from https://tinyurl.com/yc26e7va

[4] Andrea Canidio and Robin Fritsch. 2024. *Arbitrageurs' profits, LVR, and sandwich attacks: batch trading as an AMM design response*. Retrieved 2024-08-27 from https://arxiv.org/pdf/2307.02074

[5] cowswap. 2024. *Cow-amm*. Retrieved 2024-09-03 from https://cow.fi/cow-amm

[6] Arrakis Finance and Valantis Labs. 2024. *Hybrid Order Type: A new MEV aware AMM design*. Retrieved 2024-08-27 from https://github.com/ArrakisFinance/research/blob/main/HOTAMM-Whitepaper.pdf

[7] josojo. 2022. *MEV capturing AMM (McAMM)*. Retrieved 2024-08-27 from https://ethresear.ch/t/mev-capturing-amm-mcamm/13336

[8] Jason Milionis, Ciamac C. Moallemi, Tim Roughgarden, and Anthony Lee Zhang. 2024. *Automated Market Making and Loss-Versus-Rebalancing*. Retrieved 2024-09-05 from https://arxiv.org/abs/2208.06046

[9] Alex Nezlobin. 2023. *Auto-correlation strategy*. Retrieved 2024-09-05 from https://x.com/0x94305/status/1674857993740111872

[10] Mallesh Pai and Max Resnick. 2023. *Structural Advantages for Integrated Builders in MEV-Boost*. Retrieved 2024-09-05 from https://arxiv.org/abs/2311.09083

[11] Dan Robinson and Dave White. 2024. *Priority Is All You Need*. Retrieved 2024-08-27 from https://www.paradigm.xyz/2024/06/priority-is-all-you-need

[12] Sorella. 2024. *Sorella labs ethereum mev problem*. Retrieved 2024-08-27 from https://tinyurl.com/yc6a7y3r