Hybrid Order Type: A New MEV Aware AMM Design

Arrakis Finance
arrakis.fi

Valantis Labs
valantis.xyz

May 2024

Abstract

Hybrid Order Type (HOT) is a novel automated market maker (AMM) created by Arrakis Finance and Valantis Labs. The HOT AMM is designed to mitigate MEV in the form of LVR by symbiotically existing alongside intent-centric protocols. HOT pools have a hybrid execution setup whereby trades can either occur through a familiar permissionless AMM swap or through a novel RfQ system where solvers can receive a deterministic quote to trade against the pool in exchange for updating the two key AMM state variables: the spot price and the dynamic fee. The fresher pricing combined with protective dynamic fees provides robust guarantees of LVR reduction for LPs on Arrakis. In this paper, we present the core design elements of this novel AMM as well as its advantages over conventional AMMs.

1 Introduction

Automated Market Makers (AMMs) are a crucial component of decentralized finance (DeFi) protocols, enabling trustless and permissionless trading of digital assets through liquidity pools governed by algorithms. However, despite their transformative impact, existing AMM designs face several challenges that hinder their efficiency, scalability, and adaptability. Two key issues are adverse selection costs for liquidity providers (LPs) and efficient pricing mechanisms able to compete with centralized players.

One of the costs incurred by LPs is due to stale AMM prices compared to centralized exchanges. This cost is effectively quantified by the LVR metric defined in [Mil+22]. A complete taxonomy of the proposed solutions to this problem has been provided in [CR24].

On the other hand, request-for-quote (RfQ) systems have gained significant traction, since they can operate with off-chain rules that adapt to market conditions more efficiently. This can notably mitigate some of the adverse effects produced by latent blockchains while providing the benefits of traditional AMMs.

HOT AMM is a protocol designed by Arrakis Finance and Valantis Labs. It aims to provide protections against LVR extraction while maintaining desirable properties like liveliness and optimal liquidity allocation. In HOT AMM hybrid design, two modes of execution coexist, allowing retail traders to use a traditional permissionless AMM and more sophisticated solvers to get deterministic trades at competitive prices by using a special type of permissioned order. This is achieved through Valantis sovereign pools, which allow the creation of a variety of DEX designs by combining highly optimized core components and specific modules containing custom logic.

In this paper we describe the core elements of this hybrid protocol and the benefits it entails for traders and LPs.

2 An overview of Sovereign Pools

Sovereign Pools [Val24] are novel modular framework to construct decentralized exchanges (DEX). This architecture introduces a paradigm shift by embracing composability, allowing developers to assemble customized DEX designs through reusable modules. In contrast to monolithic implementations that require building each new AMM iteration from the ground up, Sovereign Pools facilitate the integration of various components responsible for functions such as pricing logic, fee calculation, oracle services, and liquidity management strategies.

A key innovation of Sovereign Pools is their ability to mitigate liquidity fragmentation, a prevalent issue in contemporary DeFi protocols, by enabling liquidity to be sourced from shared external vaults accessible to multiple pools. Furthermore, they natively support rebase tokens, a flourishing asset class characterized by dynamically adjusting supply and impulsed by liquid staking protocols. Other notable features include fee modules that facilitate dynamic and custom swap fee structures, verifier modules for permissioned or permissionless pool access, integrated decentralized oracle modules, and compatibility with liquidity mining incentive schemes. By embracing modularity and composability, Sovereign Pools empower developers to innovate on AMM designs while leveraging battle-tested components, reducing implementation complexity, and promoting security through independent module audits.

3 Flash Orders

HOT is a module in Sovereign Pools that allows external solvers/market makers to obtain signed ***Flash*** quotes from the Arrakis Quoting Service, which contain a trading volume, price, and AMM state updates like new spot price and fee parameters. These signed and time-limited quotes can then be submitted on-chain by solvers to trade against the pool's liquidity at the specified terms, with benefits such as deterministic liquidity for solvers, extra volume for liquidity providers, dynamic fees to protect against stale prices, and reduced exposure to pricing risks like top-of-block arbitrage.

HOT pools have two execution modes: permissioned Flash swaps and permissionless AMM swaps. Solvers submit orders to the Arrakis Quoting Service API and obtain signed Flash quotes. The Quoting Service can include extra parameters into these quotes, such as AMM spot price or fee updates based on volatility and other factors. Solvers can then use these signed quotes to perform a Flash swap that makes use of the pool's liquidity. After this swap is processed spot price and fee parameters are reset to the quoted values, see Section 6.1.

The Quoting Service is a trusted agent that exposes an API directly to Solvers and decides which orders to sign. The use of an internal price oracle and maximum trading

volumes provide on-chain security guarantees against malicious quotes. In the presence of a malfunctioning Quoting Service, HOT AMM hybrid mode assures continuous liveliness and a dynamic fee protects LPs against LVR extraction by arbitrageurs.

3.1 Benefits

Among the benefits of this design, we can list the following

- Solvers integrate with a deterministic source of liquidity, providing competitive pricing net gas. This source of liquidity offers a fixed price and partially fillable order for a reasonable amount of time.
- LPs on Arrakis receive extra non-toxic volume from filling Solvers directly while reducing exposure to Top-of-Block arbitrage.
- Users get better price execution, since solvers are incentivized to land Flash swaps by competing on the best price update for the HOT AMM.
- In traditional AMMs LPs use the trading fee as a mechanism of LVR mitigation, see [MMR23]. By reducing toxic arbitrage, HOT design also allows to reduce trading fees without affecting LP profitability.
- Any actor can execute low-gas AMM swaps at a minimum fee if the signed Flash quote quickly lands on-chain.
- Solvers can submit more than one Flash swap per block without worrying about reverted transactions, except in rare edge cases where the wrong input data is signed or the proposed price is significantly different from a reference on-chain Oracle.

3.2 Flash Order Type struct parameters

The Flash Order Type struct is defined in the smart contract as follows:

```
struct FlashOrderType {
    uint256 amountInMax;
    uint256 solverPriceX192Discounted;
    uint256 solverPriceX192Base;
    uint160 sqrtSpotPriceX96New;
    address authorizedSender;
    address authorizedRecipient;
    uint32 signatureTimestamp;
    uint32 expiry;
    uint16 feeMinToken0;
    uint16 feeMaxToken0;
    uint16 feeGrowthInPipsToken0;
    uint16 feeMinToken1;
    uint16 feeMaxToken1;
    uint16 feeGrowthInPipsToken1;
    uint8 nonce;
    uint8 expectedFlag;
    bool isZeroToOne;
}
```

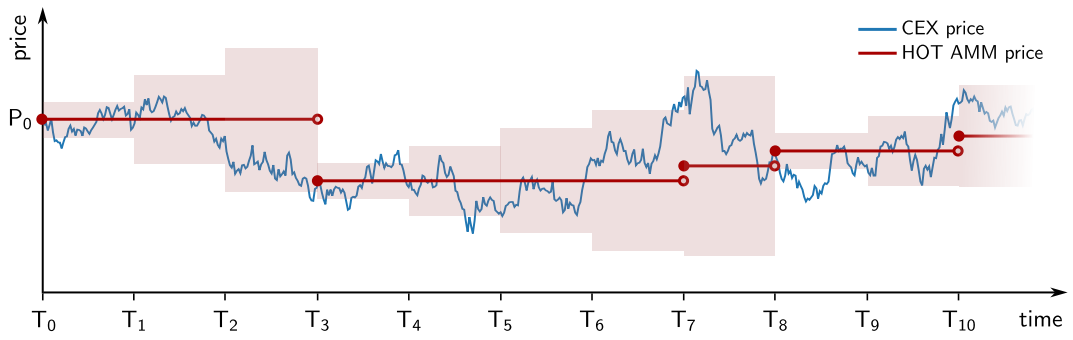
The HTTP response returned by the Quoting Service API to the solver will include a pool-specific payload containing `FlashOrderType` and an EIP-712 signature by the Quoting Service. Parameters are described as follows:

- `amountInMax`: This is the maximum amount of input token which the solver is allowed to trade against the pool. Partially fillable orders are accepted.
- `solverPriceX192Discounted`: This is the improved price that the solver gets if the Flash order includes a valid AMM state update. This price is only effective to the first solver that lands in the block.
- `solverPriceX192Base`: This is the fallback price that the solver gets if its Flash swap has been successfully validated, but did not meet the criteria to update the AMM's state variables. To simplify the price dynamics and keep it deterministic, the Quoting Service can set it equal to `solverPriceX192Discounted`, thus giving the price discount to the solver whether or not it updates the AMM state.
- `sqrtSpotPriceX96New`: The most up to date AMM square-root spot price, as specified by the Quoting Service. After the Flash order has been successfully validated, the AMM's spot price will be updated towards this value.
- `authorizedSender`: Address of `msg.sender` who is allowed to call the Valantis pool with this payload. This flag in order to prevent front-running or other kinds of unwanted accesses.
- `authorizedRecipient`: Address that must receive output token amounts after the AMM swap.
- `signatureTimestamp`: Off-chain UNIX timestamp at which the Quoting Service signed the Flash quote.
- `expiry`: Duration the Flash quote is valid. It becomes invalid once `blockTimestamp > signatureTimestamp + expiry`.
- `feeMinToken0`: Minimum AMM swap fee for swaps where token0 is the input token.
- `feeMaxToken0`: Maximum AMM swap fee for swaps where token0 is the input token.
- `feeGrowthInPipsToken0`: Growth rate per second, in percentage points, to interpolate between `feeMinToken0` and `feeMaxToken0` .
- `feeMinToken1`: Minimum AMM swap fee for swaps where token1 is the input token.
- `feeMaxToken1`: Maximum AMM swap fee for swaps where token1 is the input token.
- `feeGrowthInPipsToken1`: Growth rate per second, in percentage points, to interpolate between `feeMinToken1` and `feeMaxToken1` .
- `nonce`: Nonce used for replay protection, set as a bitmap on a `uint8` value.
- `expectedFlag`: Expected value for bit at a certain position in a nonce bitmap. See (nonce explanation docs) for more details.
- `isZeroToOne`: Direction of the swap for which the Flash swap is valid.

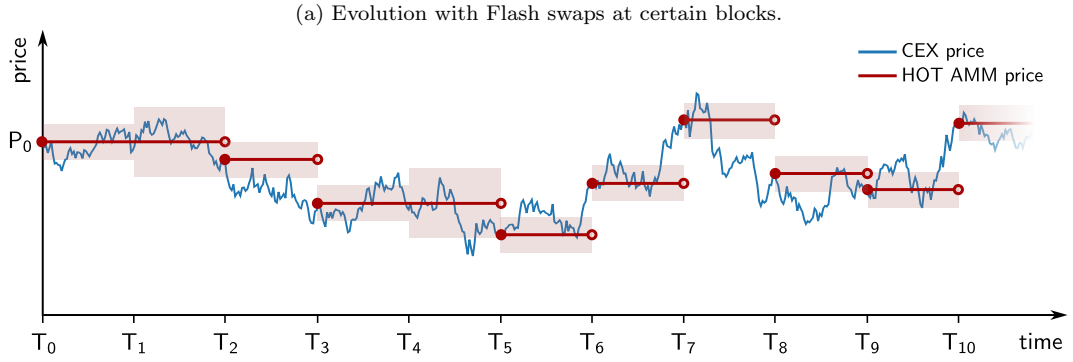
4 Dynamic Fee

Flash quotes include `feeGrowthInPipsToken{0,1}` as parameters. This is the growth, in pips per second, of the fee percentage applied to AMM swaps. These parameters can be updated on-chain by a Flash Swap if the Quoting service includes it in the payload. The AMM swap fee for each input token is given by

$$\texttt{fee\_bips\_token}\{0,1\} = \texttt{feeMinToken}\{0,1\} + \texttt{feeGrowthInPipsToken}\{0,1\} \times (\texttt{blockTimestamp} - \texttt{lastProcessedSignatureTimestamp})/1000000$$



(a) Evolution with Flash swaps at certain blocks.



(b) Evolution with Flash swaps at every block.

Figure 1: Price and dynamic fee evolution on a HOT pool in the unique presence of arbitrageurs. The pale red zones represent the no-trade regions for stale-price arbitrageurs and are determined by the dynamic fee. Notice that stale-price arbitrage, for example at times T_7 and T_{10} in 1a, becomes less profitable with time. On the contrary, Flash swaps, see 1b, keeps the HOT AMM with up to date prices and updates the no-trade region neutralizing any subsequent arbitrages.

Where `lastProcessedSignatureTimestamp` is the UNIX timestamp at which the previously processed Flash quote has been signed (set off-chain, but validated to be within reasonable bounds by the smart contract).

AMM swap fees are capped at their respective maximum value (`feeToken0Max` or `feeToken1Max`).

4.1 Dynamic fee preventing stale price arbitrage

The above shows a fee which grows linearly as the last processed AMM state update becomes stale, see Figure 1a.

`feeGrowthInPipsToken{0,1}` becomes the expected per-second change in spot price, in percentage points. Arrakis Quoting Service can update this parameter to reflect the current expected volatility of an asset pair across external markets. Therefore the AMM spot price + fee should be sufficient to prevent arbitrage opened up by large block times, which might cause the Flash quotes to land on-chain later than expected, and would therefore update the AMM towards a stale price.

This dynamic fee also ensures AMM swaps occurring directly after a Flash quote is processed receive a minimum fee. The minimum fee can be reasonable lower than comparable AMMs, because a recently signed AMM spot price reduced the probability of stale-price arbitrage.

This dynamic fee allows the pool to protect itself from Top-of-Block arbitrage without applying an explicit lock on permissionless swaps. In practice, the AMM swap fee growth should outpace the divergence between the "true" market price (say, from a CEX) and the AMM's spot price, to minimize exposure to toxic order-flow. It is up to the Quote Service to constantly update these parameters in response to changing market conditions such as volatility. When Flash quotes do not land in the next block, it is still possible to move the AMM spot price, but the fee will account for the fact that the signed price may be stale and applies a fee spread to protect against this probability, see Figure 1a.

As a protection measure, the smart contract enforces that `feeToken0Min` and `feeToken1Min` cannot be below some predetermined value (`minAmmFee`). This is a minimum error assumed to the signer's prices, which reduces the chance of arbitrages after AMM updates. Also, these fees determine a lower bound on the cost incurred by noise traders for the right to access the active liquidity in the pool.

5 Effective AMM Liquidity

HOT AMM assumes that at any point in time the reserves can be in any arbitrary composition. This is because incoming Flash swaps can trade at any price (restricted by some configurable bounds). So, the fundamental invariants of CPMMs do not hold for the reserves.

To make sure that the AMM works correctly under any circumstance the reserve composition looks like, we decouple the reserves into 2 categories:

1. **Active Reserves** These are balanced reserves that adhere to a translated $xy = k$ invariant [Ada+21, Section 2], and can be utilized by both the AMM and Flash swaps.

2. **Passive Reserves** Any unbalanced reserves that are not part of the active reserves are considered passive. Only Flash swaps can access the passive reserves.

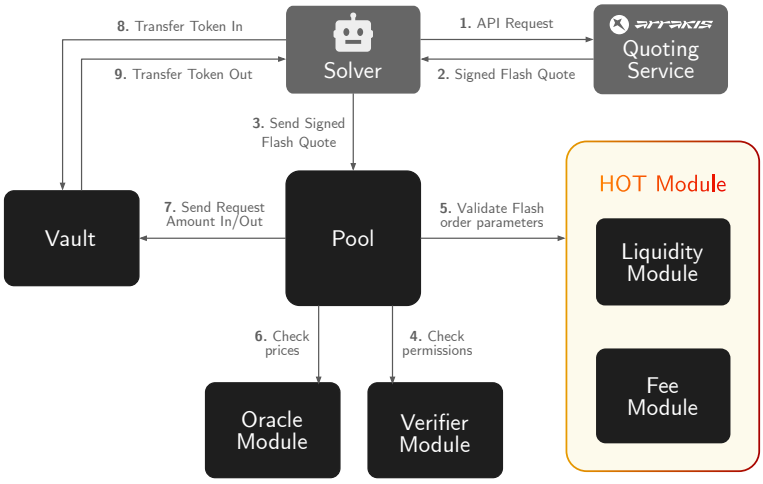


Figure 2: Flash swap flow.

6 Swaps

In this section, we describe in detail the two modes of execution for swaps. First, we explain the flow of Flash swaps. These permissioned swaps can in practice only be accessed by highly specialized actors like searchers/solvers that run complex mathematical algorithms to find profitable trades. Second, we briefly describe how the uninformed traders interact with this protocol via a conventional and permissionless concentrated liquidity AMM.

6.1 Flash Swap

A Flash swap begins when an authorized solver (market maker) calls the Sovereign Pool contract with a signed Flash order quote payload obtained from the pool's Quoting Service API, see Section 3.2. The pool first performs basic validations on this data, checking aspects like expiry, amount limits, address authorizations, and price bounds. It then queries the HOT contract, which implements both the Liquidity Module and Swap Fee Module interfaces, passing along the quote data and any verification context.

Within the HOT contract, it first authenticates that the swap fee quote path was chosen correctly by the sender. It then enforces limits like maximum quotes per block, validates the quote parameters against configured thresholds (minimum fees, price deviations from oracle, etc.), and verifies the Quoting Service signature on the quote data. If all checks pass, the Flash swap updates its internal state with the new spot price, fee growth rates, and other parameters from the signed quote. It also determines the final price to quote to the recipient based on whether the AMM state will be updated. Finally, execution returns to the pool contract to complete the swap using the quoted amounts, price, and fees, while transferring tokens and updating reserves accordingly.

6.2 AMM Swap

Retail traders can interact with no restrictions with an AMM that holds the active reserves of the vault. One can think of this AMM as a single standard Uniswap V3 range order [Ada+21] with a unique liquidity provider, where `sqrtPriceLowX96` and `sqrtPriceHighX96` represent the upper and lower bounds of the range. This range is determined and actively managed by the `liquidityProvider`, to provide the best risk-adjusted returns to its users. In the current implementation, this role is taken by an Arrakis Vault, which tokenizes the range position to make it accessible to other LPs. In addition, despite having dynamic fees that evolve with time (see Section 4), AMM swaps benefit from the hybrid nature of HOT AMM, since Flash swaps maintain the trading price up to date. Finally, this swap mode enables composability with other DeFi applications as well as making HOT AMM accesible to anyone.

References

[Ada+21] Hayden Adams et al. "Uniswap v3 core". In: *Tech. rep., Uniswap, Tech. Rep.* (2021).

[CR24] Daniel Contreras and Ari Rodríguez. *LVR from Theory to Practice: A Survey of Next Generation DEX Designs*. 2024. URL: https://github.com/ArrakisFinance/research/blob/main/LVR-from-Theory-to-Practice.pdf (visited on 05/09/2024).

[MMR23] Jason Milionis, Ciamac C. Moallemi, and Tim Roughgarden. "The Effect of Trading Fees on Arbitrage Profits in Automated Market Makers". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2023, pp. 262–265.

[Mil+22] Jason Milionis et al. "Automated market making and loss-versus-rebalancing". In: *arXiv preprint arXiv:2208.06046* (2022).

[Val24] Valantis Labs. *Sovereign Pool Documentation*. 2024. (Visited on 05/08/2024).

Disclaimer

This paper serves for general information purposes only and does not constitute investment advice. The opinions reflected herein are the authors' current viewpoints, not made on behalf of Arrakis Finance or Valantis Labs. These opinions are subject to change without being updated. Readers should not rely on this paper for accounting, legal or tax advice, investment recommendations, or in the evaluation of making any investment decision.