Dynamic Fee Models for Mitigating Arbitrage Losses in Balancer Pools

Santeri Helminen, Antero Eloranta

Abstract

This study examines the efficacy of both static and dynamic fee functions in automated market makers across Ethereum mainnet, Arbitrum, and Polygon. We analyze fee structures based on static fees, volatility, DEX trading volume, and gas prices for eight token pairs. Our aim is to improve pool profitability by capturing a larger portion of CEX-DEX arbitrage opportunities.

Using historical data from August 2023 to June 2024, we recreate pool states at each block and compare them with Binance prices to quantify arbitrage opportunities. Our findings reveal that the effectiveness of fee structures varies significantly across pools and networks. Notably, optimizing static fees often leads to substantial improvements, sometimes outperforming dynamic models. Among dynamic approaches, volume-based functions showed consistent potential for improvement, while gas price-based functions were particularly effective on the Ethereum mainnet. Networks with shorter block times demonstrated fewer opportunities for improvement through fee adjustments.

Results indicate that AMM fee structures should be tailored to each pool's specific characteristics, considering factors such as underlying blockchain, market conditions, and trading patterns. In some cases, simply adjusting the static fee can yield significant benefits, while in others, implementing dynamic fee models provides superior results. This research provides insights for AMM protocol designers and liquidity providers, offering a framework for implementing more efficient and adaptive fee structures in decentralized finance.

Table of Contents

1. Introduction

For most assets traded on on-chain automated market makers (AMMs) such as Balancer the price discovery process happens in off-chain venues. As time in blockchains is discrete this leads to situations where arbitrageurs can take advantage of stale AMM prices and profit by arbitraging the price difference between off-chain and on-chain venues. As trading is a zero-sum game this information asymmetry between AMM liquidity providers (LPs) and arbitrageurs results in a profit made by informed arbitrageurs to incur a loss of the same size as LPs.

This introduced the problem of how to select optimal AMM fees. On one hand, if the fee is set too high neither arbitrageurs nor noise traders want to interact with the pool leading to limited fee collection and AMM price that is far away from the off-chain venue's price. On the other hand, if the fee is set too low arbitrageurs will constantly interact with the pool generating huge amounts of volume. However, if the fee is too low, fee collection will be limited despite the huge volume.

We approach the problem by studying different fee function candidates by trying to optimize for the maximum amount of value LPs can get from the CEX DEX arbitrage situations. This means that we try to find fee functions that optimize the trade-off between incentivizing arbitrageurs to trade on a pool and the percentage of arbitrage collected by LPs by maximizing fees collected by LPs.

2. Data and methodology

2.1. Data

To calculate CEX DEX arbitrage, we need access to the AMM price and a price from the venue where price discovery happens. Being the most liquid venue, we follow the Binance price as our off-chain venue price. Using the event data from Balancer pools and Balancer Vault's historical pool balance eth calls, we recreate the pool state at each block and merge it with the respective Binance price using the block timestamp.

We collect data for the following Balancer pools on top of the Ethereum mainnet, Arbitrum One, and Polygon PoS:

Mainnet pools: BAL-WETH, GNO-WETH, WBTC-WETH, STG-USDC, RNDT-WETH
Arbitrum pools: STG-USDC.e, RDNT-WETH
Polygon pools: GHST-USDC

These pools were selected to cover a variety of different types of pools ranging from big to small and continuously traded extremely liquid pairs to less liquid pairs. In addition, we analyze pools on top of different blockchains to account for differences in block times.

Dataset blocks

Pool	**Blocks**	**Dates**
BAL-WETH Mainnet	18000000 - 20000000	Aug 26, 2023 - Jun 2, 2024
GNO-WETH Mainnet	18000000 - 20000000	Aug 26, 2023 - Jun 2, 2024
WBTC-WETH Mainnet	18000000 - 20000000	Aug 26, 2023 - Jun 2, 2024
STG-USDC Mainnet	18000000 - 20000000	Aug 26, 2023 - Jun 2, 2024
RNDT-WETH Mainnet	18466412 - 20000000	Oct 31, 2023 - Jun 2, 2024
STG-USDC.e Arbitrum	71246324 - 217413094	Mar 18, 2023 - Jun 1, 2024
RDNT-WETH Arbitrum	71246324 - 217413094	Mar 18, 2023 - Jun 1, 2024
GHST-USDC Polygon	46783038 - 57659264	Aug 26, 2023 - Jun 1, 2024

There is a gap in the dataset for all pools from March 9, 2024, to March 31, 2024, due to the unavailability of Binance price data during this period.

We collect historical 1-second candle data from Binance for the same period. As the most liquid and actively traded pair on Binance for Ethereum is Ethereum - USDT, we use USDT for dollar price calculations. For pairs that have WETH as one of the assets, we calculate the implied Binance prices based on 1-second candle data of ETH - USDT combined with the

corresponding USDT pair on Binance. For example, we calculate the implied Binance price for BAL - WETH based on BAL - USDT and ETH - USDT pairs.

2.2. Calculating arbitrage

To quantify CEX DEX arbitrage in each block, we compare AMM's price at the end of the previous block and compare it with Binance's price at the time of the next block. Assuming arbitrageurs are rational they want to move the price to a point where they would receive the same amount of assets from both AMM and Binance for the next marginal unit they trade. This point P^* can be expressed as:

$P^* = \frac{P_B}{1+f}\ if\ P_{AMM} < P_B$

$P^* = P_B * (1 + f)\ if\ P_{AMM} > P_B$.

We quantify CEX DEX arbitrage by calculating what is the average price for a transaction that moves the AMM price P^* and the amount of assets swapped. This can be determined from AMM balances.

Balancer determines value function consisting of pool's weights and balances to be constant

$V = \prod_t B_t^{W_t}$

where t is the number of tokens in the pool, B is the balance of a token and W is the weight of the token.

The spot price of a token pair is determined by pool weights as $SP_i^o = \frac{\frac{B_i}{W_i}}{\frac{B_o}{W_o}}$ and after a swap happens $SP_i^{o'} = \frac{\frac{B_i - \Delta B_i}{W_i}}{\frac{B_o + \Delta B_o}{W_o}}$.

When a swap happens $B_i^{W_i} * B_o^{W_o} = V = (B_i - \Delta B_i)^{W_i} * (B_o + \Delta B_o)^{W_o}$ must hold, excluding AMM fees. The same can be expressed as
$W_i * ln(B_i) + W_o * ln(B_o) = W_i * ln(B_i - \Delta B_i) + W_o * ln(B_o + \Delta B_o)$.
Differentiating with respect to ΔB_i: $\frac{\partial \Delta B_o}{\partial \Delta B_i} * \frac{W_o}{B_o + \Delta B_o} - \frac{W_i}{B_i - \Delta B_i} = 0$. Which can be rearranged as $\frac{\partial \Delta B_o}{\partial \Delta B_i} = \frac{W_i}{W_o} * \frac{B_o + \Delta B_o}{B_i - \Delta B_i}$.

Knowing this we can determine the average price $P_a = \frac{1}{\Delta B_i} * \int_0^{\Delta B_i} \left[\frac{W_i}{W_o} * \frac{B_o + x}{B_i - x} \right] \partial x$.

This can be approximated to be $P_a \approx \frac{W_i}{W_o} * \frac{B_o - \frac{\Delta B_o}{2}}{B_i - \frac{\Delta B_i}{2}}$.

This can be expressed in terms of spot prices $P_a \approx \sqrt{SP_i^o * SP_i^{o'}}$.

By knowing the Binance price and the average price of the swap moving the AMM price P^* we can quantify CEX DEX arbitrage $\left| P_b - P_a \right| \Delta B_i$.

CEX/DEX Arbitrage Calculations
Blocks where the pool has one or more swaps. Fees and arbitrage value have been calculated based on the theoretical maximum arbitrage available for said blocks. The calculations do not take gas fees into account.

To ensure the reliability of our results and handle outliers effectively, the calculations filter out the highest decile of rows based on the price difference between the pool and Binance. This approach helps us handle outliers and obtain more reliable results by excluding extreme price discrepancies that may not represent typical arbitrage opportunities.

For all LVR calculations, the token reserve weights have been normalized to 0.5 / 0.5 with the following formula: $reserve \ * \ (0.5 \ / \ weight)$

Pool	**Average fee %**	**Blocks with arbitrage %**	**Fees from arbitrage value %**
BAL-WETH	0.75	30.86	37.08
GNO-WETH	0.26	81.03	7.57
WBTC-WETH	0.22	18.07	42.32
STG-USDC (Mainnet)	0.67	39.24	12.27
RNDT-WETH (Mainnet)	0.50	35.12	7.92
STG-USDC.e (Arbitrum)	0.30	3.19	54.78
RDNT-WETH (Arbitrum)	0.50	0.00	0.00
GHST-USDC (Polygon)	0.30	68.86	35.92

2.3. Fee formulas

Input	Source	Chain
1. Static fee	-	Mainnet, Arbitrum, Polygon
2. Volatility	DEX	Mainnet, Arbitrum, Polygon
3. DEX Volume	DEX	Mainnet, Arbitrum, Polygon
4. Gas Price	Chain block information	Mainnet, Polygon

1. Static fee range between 0.1% - 1.0%

2. The volatility measure used in this analysis is calculated as the coefficient of variation of price within each hour:

Volatility = Standard Deviation of Price / Average Price

This ratio expresses price variability relative to the average price. Higher values indicate greater price instability within the time period. It expresses the standard deviation as a percentage of the mean, giving a dimensionless number that allows for comparison between datasets with different units or means.

3. Hourly DEX trading volume normalized by pool token reserves

4. Hourly average of gas fees

We backtest the following fee functions using the above input variables:

Function	Input
Input variable	Volatility, DEX Volume, Gas Price
Logarithmic input variable	Volatility, DEX Volume, Gas Price
Sigmoid input variable	Volatility, DEX Volume, Gas Price
Input variable variance	DEX Volume, Gas Price
Mean input variable	DEX Volume, Gas Price
Square root input variable	DEX Volume, Gas Price
Exponential input variable	Volatility, Gas Price
Quadratic input variable	Volatility

3. Results

In this section, we analyze the performance of various dynamic fee functions across different token pairs and blockchain networks. Our goal is to identify fee structures that can potentially improve pool profitability while maintaining or reducing average fees. We evaluate each function's ability to capture arbitrage opportunities and generate fees compared to the current fee model.

For each pool, we set the parameters of our dynamic fee algorithms to achieve approximately the same average fee as the current fee percentage of the pool. This approach allows for a fair comparison between the current fee model and the proposed dynamic fee functions. We examine how these functions respond to changes in gas prices, market volatility, and trading volume, and assess their impact on total fees collected and average fee rates.

In addition to dynamic fee models, we perform calculations using a range of static fees. This provides a baseline for comparison and helps explore how the fee level itself affects fee earnings. By including both dynamic and static fee calculations, we can better understand the potential benefits of implementing dynamic fee structures and identify scenarios where adjusting the static fee might suffice to improve pool performance.

3.1. BAL - WETH on mainnet

Figures 1, 2, 3 and 4 illustrate different dynamic fee functions' changes in collected fees for the BAL-WETH pool based on static, gas, volatility, and volume, respectively. Pink lines represent the amount of fees the pool currently collects while green lines represent the current fees charged by the pool. Values above the pink line have higher profits than what the pool is currently getting while values below the line have lower profits. Values left from the green line represent lower average pool fees while values right from the green line represent higher average fees. Optimal values are the ones that are above the pink line while also being left on the green line. Functions that can achieve such values are able to collect more fees than the current fee function while also lowering the average fee charged by the pool.

As visualized in Figure 1 the collected fees can be increased by decreasing the average fees. The collected fees reach the optimal point at around 0.50% corresponding to an increase of about 40%.

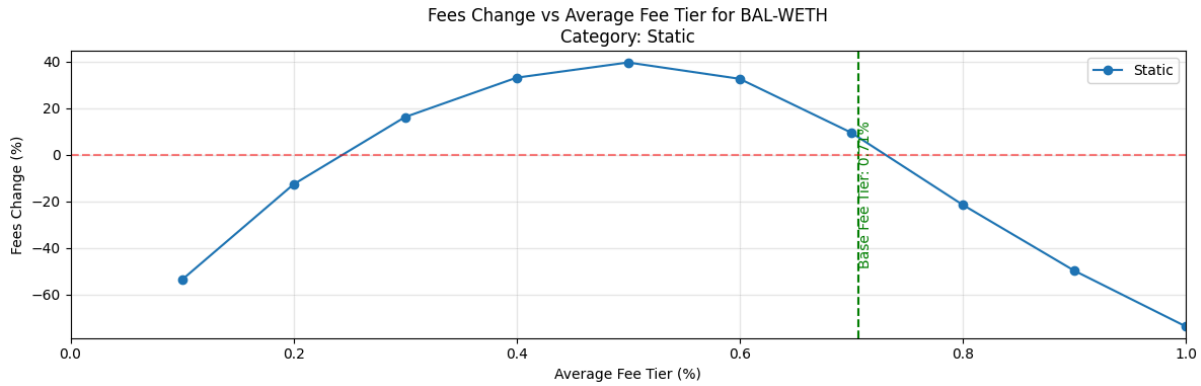

Figure 1: Static fee for BAL - WETH

As visualized in Figure 2 out of gas-based functions the ones that take gas as a parameter to exponential, square root, sigmoid, and linear functions have higher profit and lower average fees than the current fee function. Out of these functions logarithmic and square root functions are the best ones increasing the profit by 30-40% while decreasing the average fee significantly.

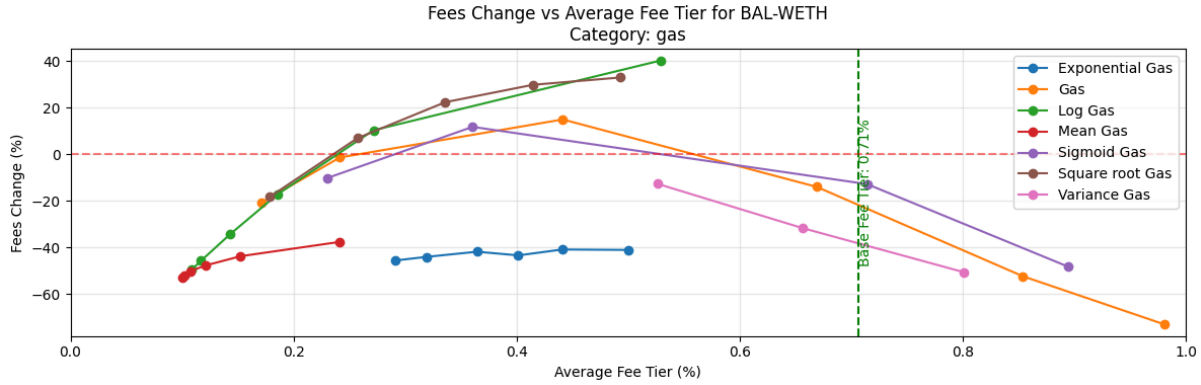

Figure 2: Change in collected fees and average fees for gas-based BAL - WETH functions

As visualized in Figure 3 out of volatility-based functions the ones that take volatility as a parameter all functions except the exponential function have higher profit and lower average fees than the current fee function. Out of these linear and quadratic functions are the best ones increasing the profit by 40% while decreasing the average fee the most.

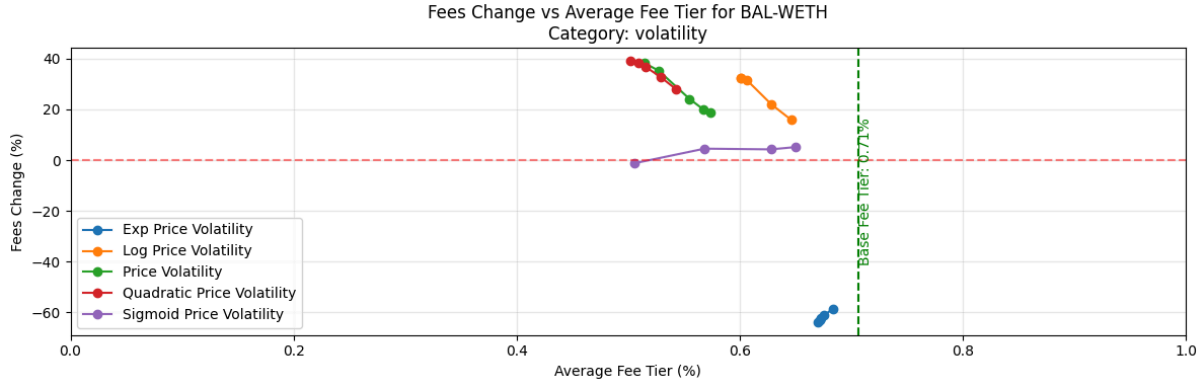

Figure 3: Change in collected fees and average fees for volatility-based BAL - WETH functions

As visualized in Figure 4 out of volume-based functions the ones that take volume as a parameter all functions have higher profit and lower average fees than the current fee function. Out of these square root and variance functions are the best ones increasing the profit by 20-30% while decreasing the average fee the most.

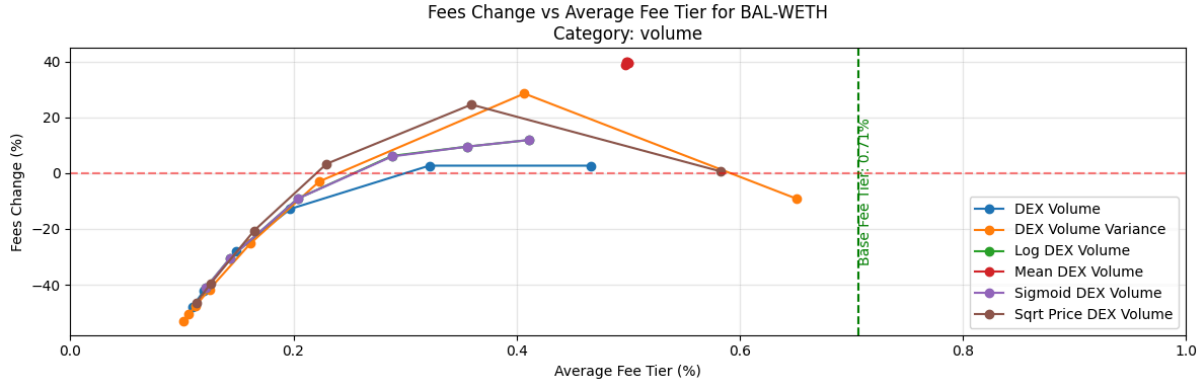

Figure 4: Change in collected fees and average fees for volume-based BAL - WETH functions

Out of all studied functions, the static and gas-based functions increase the profit the most while also decreasing the average fee the most. Gas-based functionS offer only slight improvements over static fee in certain edge cases and thus BAL-WETH could be improved the most by optimizing static fees.

3.2. GNO - WETH on mainnet

Figures 5, 6, 7, and 8 illustrate different dynamic fee functions' changes in collected fees for the GNO-WETH pool based on static, gas, volatility, and volume, respectively. As visualized in Figure 5 the collected fees cannot be increased without increasing the average fees of the pool.

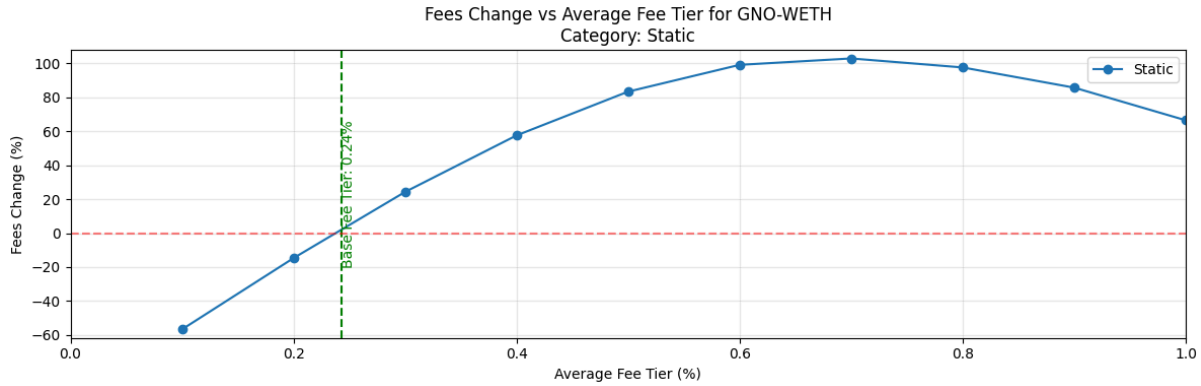

Figure 5: Static fee for GNO - WETH

As visualized in Figure 6 out of gas-based functions no function has both a higher profit and a lower average fee than the current fee function. However, all the functions that take gas as a parameter, except mean, can achieve higher profit while also having a higher average fee. Out of these functions, square root and logarithmic functions have the highest profit increase of up to 90%.

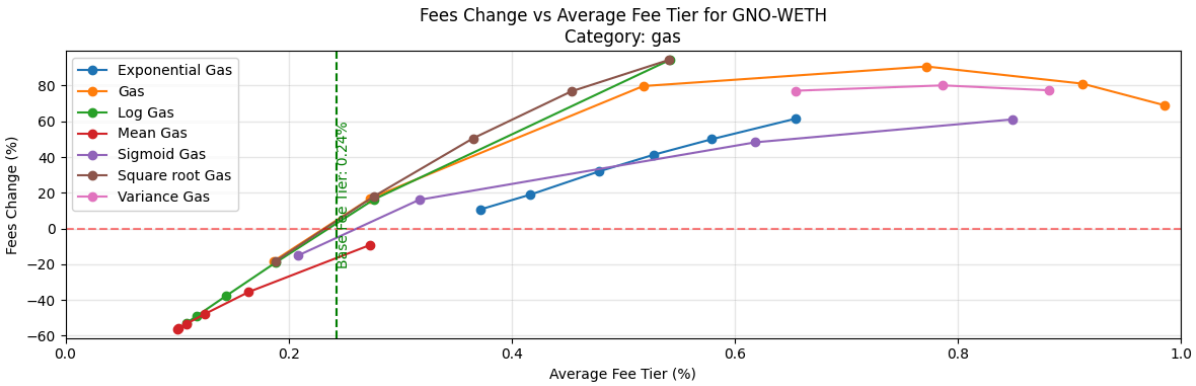

Figure 6: Change in collected fees and average fees for gas-based GNO - WETH functions

As visualized in Figure 7 out of volatility-based functions no function has both higher profit and lower average fee than the current fee function. However, exponential functions that take volatility as a parameter can achieve higher profit while also having a higher average fee. The exponential function can achieve profit increase of up to 20%.

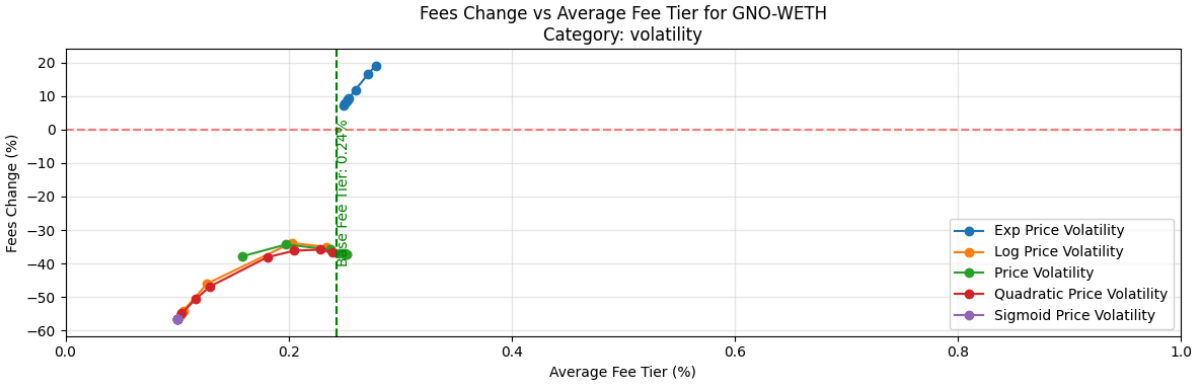

Figure 7: Change in collected fees and average fees for volatility-based GNO - WETH functions

As visualized in Figure 8 out of volume-based functions no function has both a higher profit and lower average fee than the current fee function. However, all the functions that take volume as a parameter can achieve higher profit while also having a higher average fee. Out of these functions square root function has the highest profit increase of 90%.

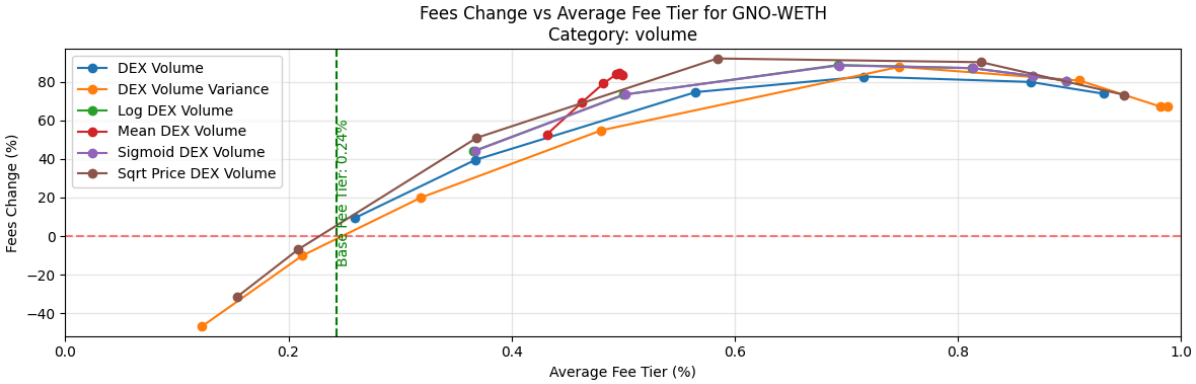

Figure 8: Change in collected fees and average fees for volume-based GNO - WETH functions

Out of all studied functions, the static function can increase the profit the most. However, if trying to optimize for a trade-off between profit increase and average fee change the gas-based functions have the best trade-off with only slightly worse profit while significantly lower average fees.

3.3. STG - USDC on mainnet

Figures 9, 10, 11, and 12 illustrate different dynamic fee functions' changes in collected fees for the STG-USDC pool based on static, gas, volatility, and volume, respectively. As visualized in Figure 9 static fees can increase the collected fees by 70% while also decreasing the average fees.

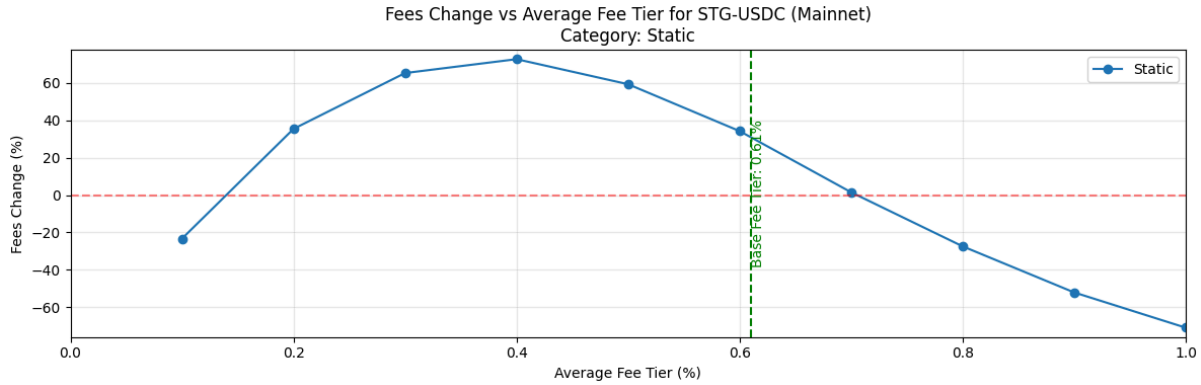

Figure 9: Static fee for STG - USDC

As visualized in Figure 10 out of gas-based functions the ones that take gas as a parameter all functions, except exponential and mean functions, have higher profit and lower average fees than the current fee function. Out of these logarithmic and square root functions are the best ones increasing the profit by 60-70%.

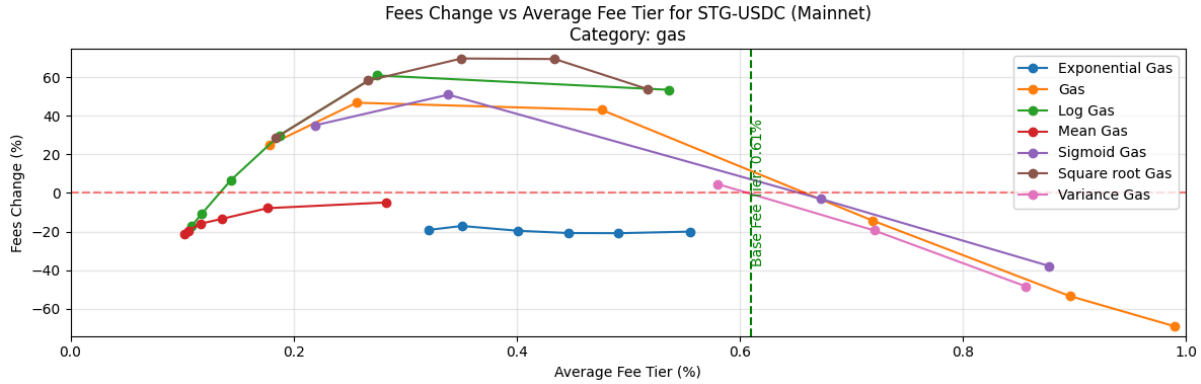

Figure 10: Change in collected fees and average fees for gas-based STG - USDC functions

As visualized in Figure 11 out of volatility-based functions the ones that take volatility as a parameter all functions except the exponential function have higher profit and lower average fees than the current fee function. Out of these quadratic function is the best and increases the profit by over 40% while decreasing the average fee.

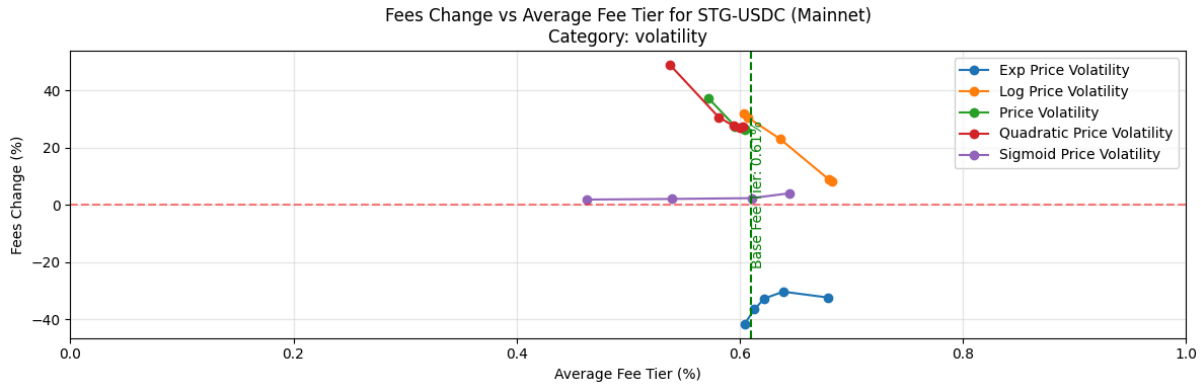

Figure 11: Change in collected fees and average fees for volatility-based STG - USDC functions

As visualized in Figure 12 out of volume-based functions the ones that take volume as a parameter all functions have higher profits and lower average fees than the current fee function. Out of these, the square root function is the best and increases the profit by 55% while decreasing the average fee.

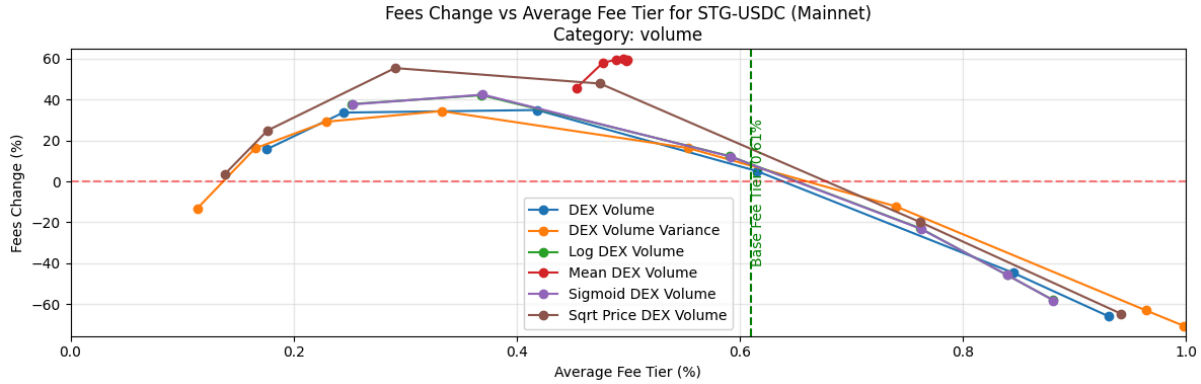

Figure 12: Change in collected fees and average fees for volume-based STG - USDC functions

Out of all studied functions, the static function can increase the profit the most. However, if trying to optimize for a trade-off between profit increase and average fee change the volume-based functions have the best trade-off with only slightly worse profit while significantly lower average fees.

3.4. RDNT - WETH on mainnet

Figures 13, 14, 15, and 16 illustrate different dynamic fee functions' changes in collected fees for the RDNT-WETH pool based on static, gas, volatility, and volume, respectively. As visualized in Figure 13 changing the static fees cannot increase the profits.

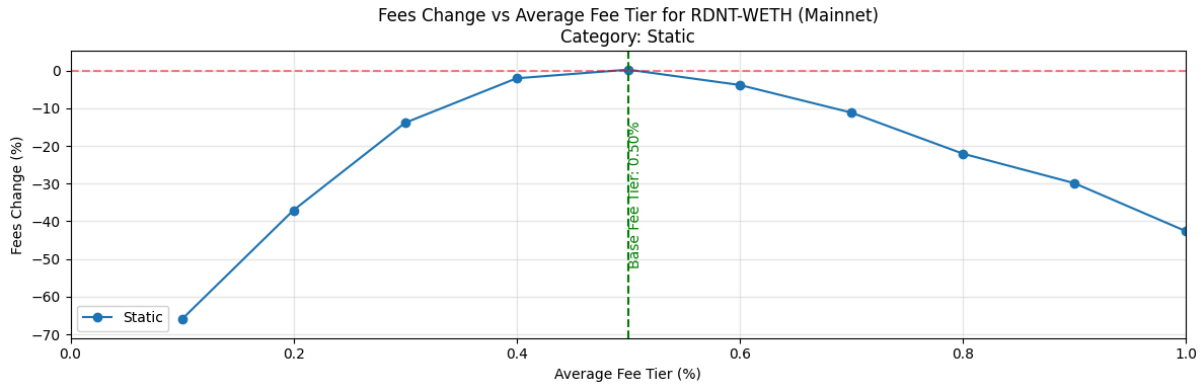

Figure 13: Static fee for RDNT - WETH

As visualized in Figure 14 no gas-based function can increase the profits.

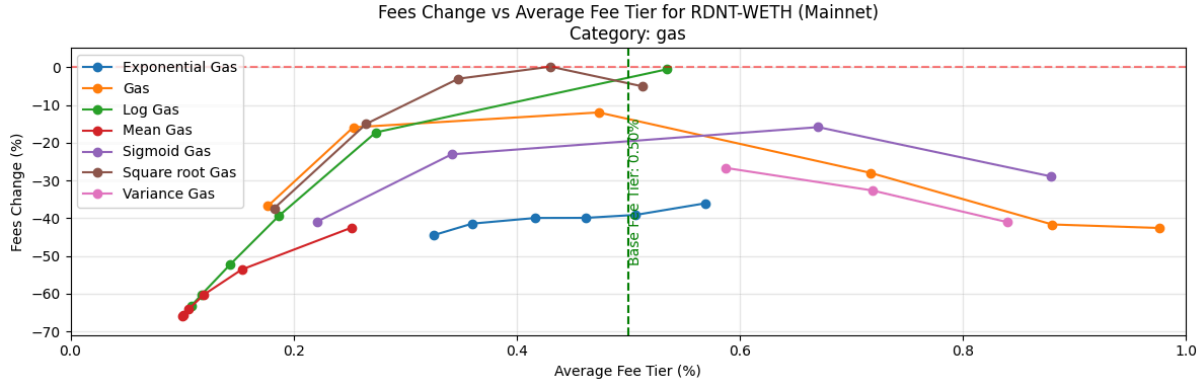

Figure 14: Change in collected fees and average fees for gas-based RDNT - WETH functions

As visualized in Figure 15 no volatility-based function can increase the profits.

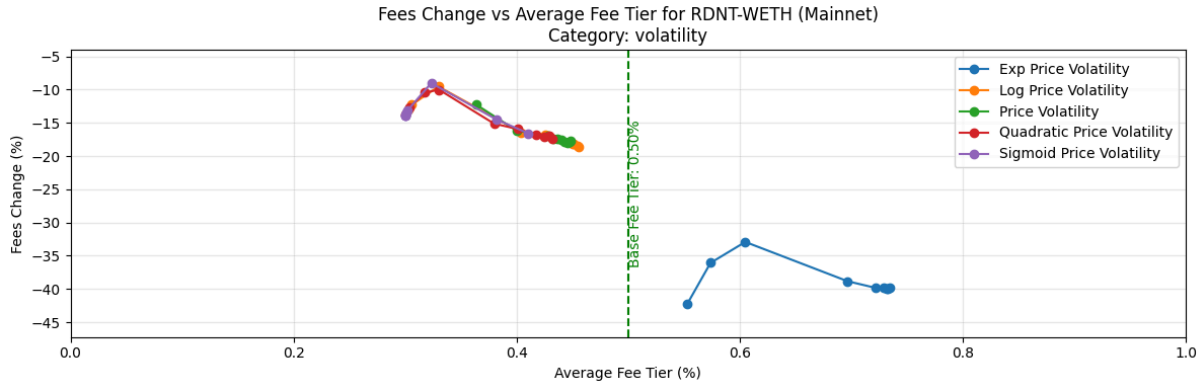

Figure 15: Change in collected fees and average fees for volatility-based RDNT - WETH functions

As visualized in Figure 16 no volatility-based function can increase the profits.

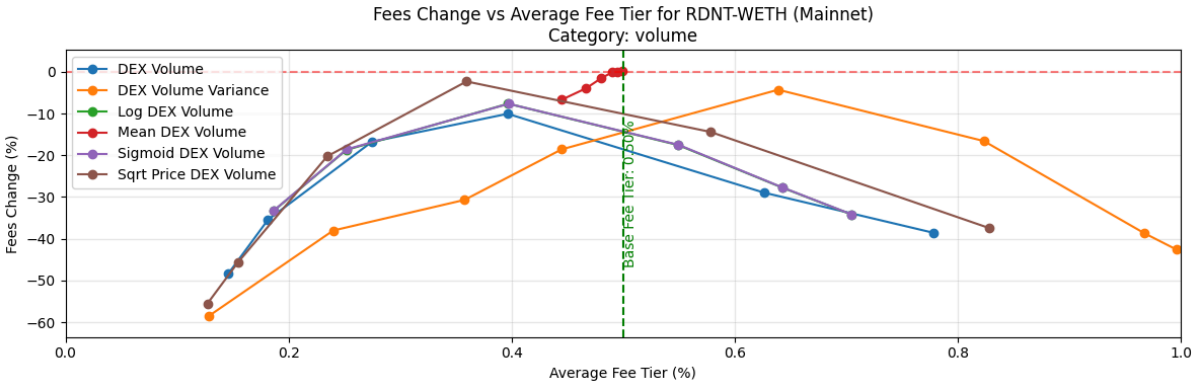

Figure 16: Change in collected fees and average fees for volume-based RDNT - WETH functions

Out of all studied functions, no function could increase the profits. Although, square root gas and mean DEX volume reached close to the base earnings with slightly lower fee percentage.

3.5. WBTC - WETH on mainnet

Figures 17, 18, 19, and 20 illustrate different dynamic fee functions' changes in collected fees for the WBTC-WETH pool based on static, gas, volatility, and volume, respectively. As visualized in Figure 17 decreasing static fees could increase the profits by 35%.

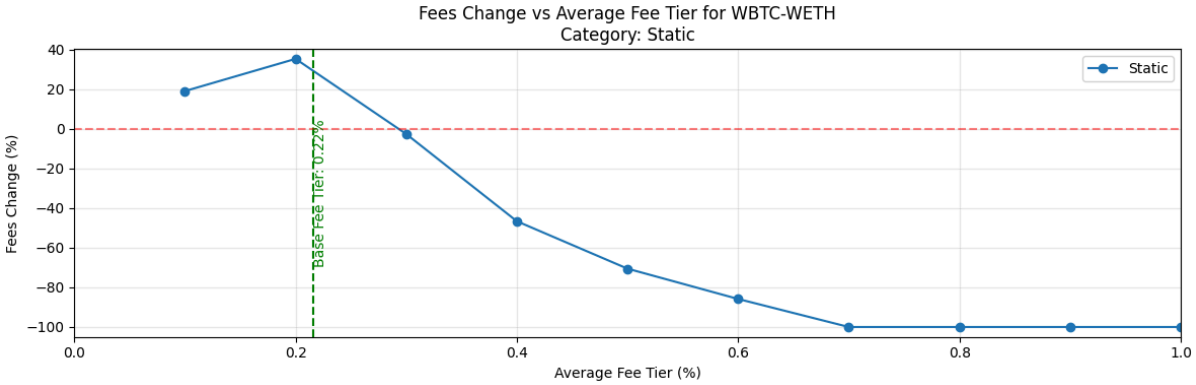

Figure 17: Static fee for WBTC - WETH

As visualized in Figure 18 out of gas-based functions half of the functions have both higher profit and lower average fee than the current fee function. Out of these, the logarithmic function is the best and increases the profit by 50% while decreasing the average fee.

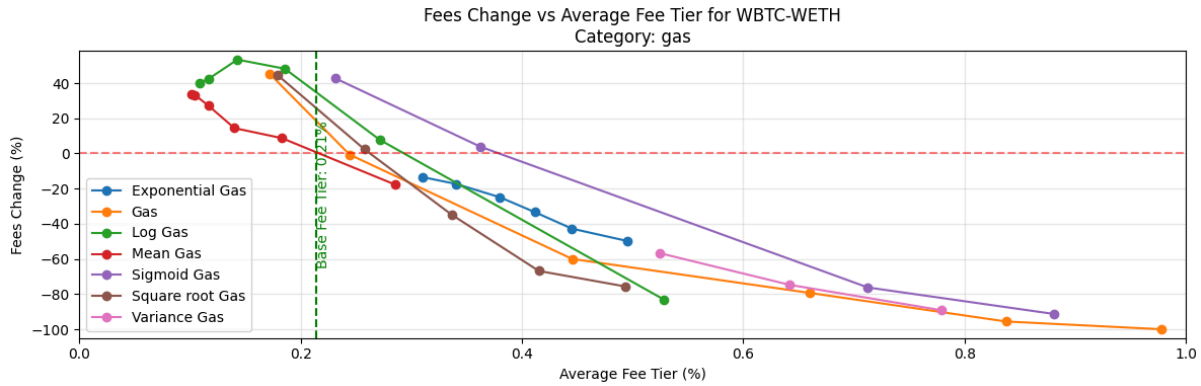

Figure 18: Change in collected fees and average fees for gas-based WBTC - WETH functions

As visualized in Figure 19 out of volatility-based functions the ones that take volatility as a parameter all functions, except exponential, have higher profit and lower average fees than the current fee function. The best functions increase the profit by 40% while not decreasing the average fee.

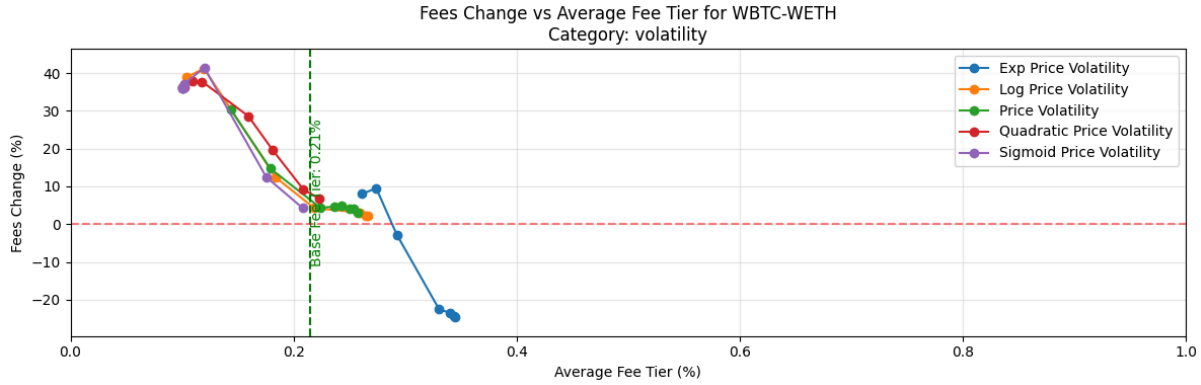

Figure 19: Change in collected fees and average fees for volatility-based WBTC - WETH functions

As visualized in Figure 20 out of volume-based functions the ones that take volume as a parameter most functions have higher profit and lower average fees than the current fee function. The best functions increase the profit by 40% while not decreasing the average fee.

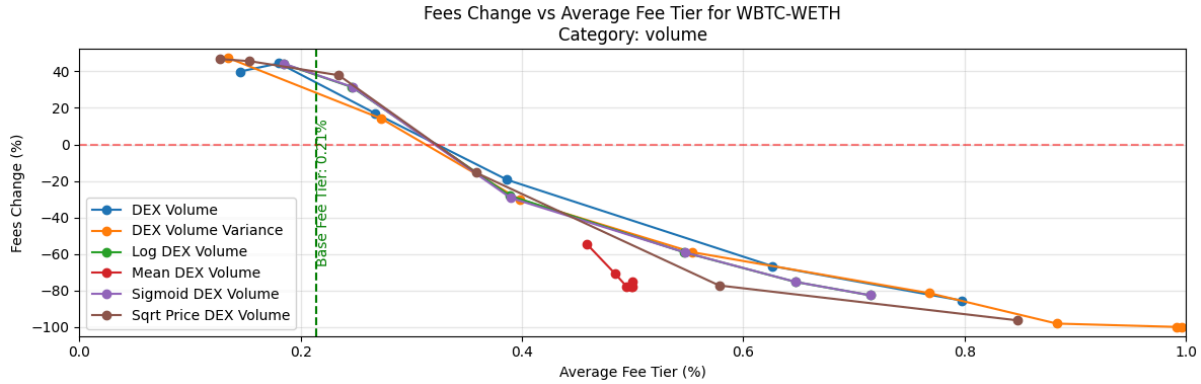

Figure 20: Change in collected fees and average fees for volume-based WBTC - WETH functions

Out of all studied functions, the volume-based functions can increase the profit the most.

Both volume and gas input variables have their optimal fee tiers near the base fee tier. Volume-based fee functions show the highest potential at 40% higher total fees compared to the base at around the same average fee percentage.

3.6. STG - USDC on Arbitrum

Figures 21, 22 and 23 illustrate different dynamic fee functions' changes in collected fees for the STG-USDC pool based on static, volatility and volume, respectively. As visualized in Figure 21 decreasing static fees could increase the profits by 45%.

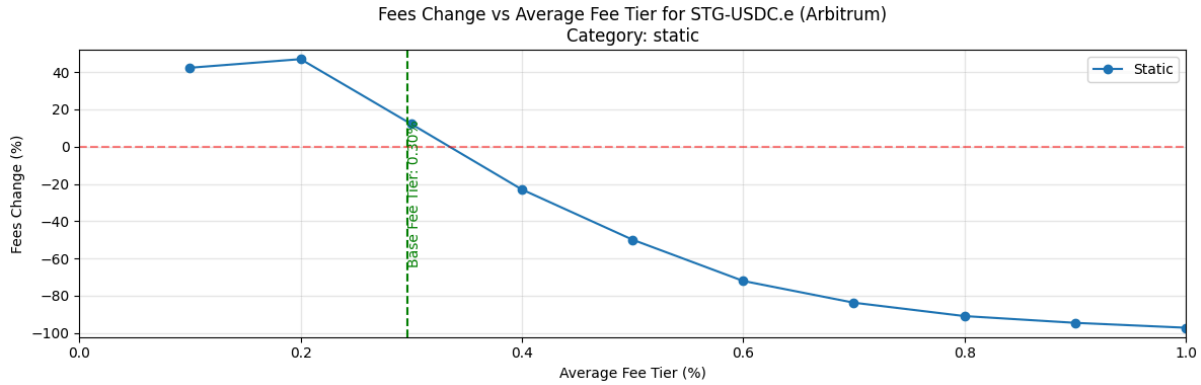

Figure 21: Static fee for STG - USDC

As visualized in Figure 22 out of volatility-based functions only sigmoid has higher profit and lower average fee than the current fee function.

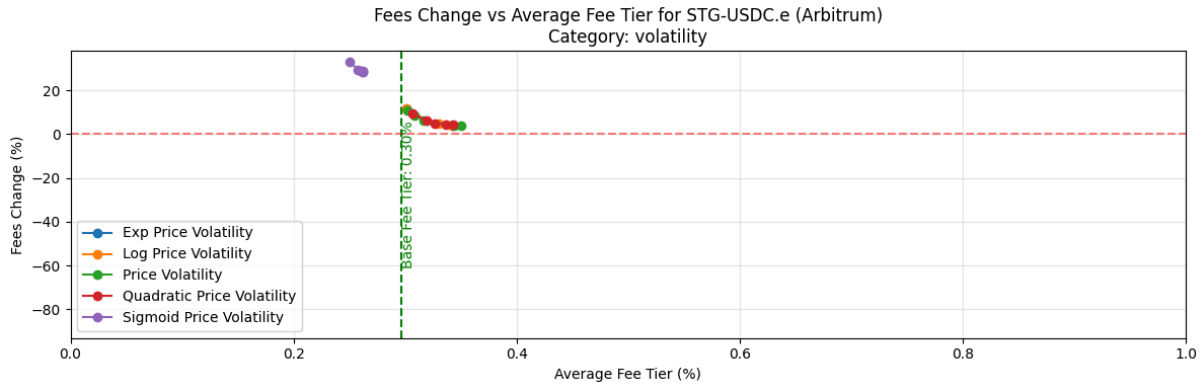

Figure 22: Change in collected fees and average fees for volatility-based STG - USDC functions

As visualized in Figure 23 out of volume-based functions all except the mean function have higher profits and lower average fees than the current fee function. The best functions have 60% higher collected fees.

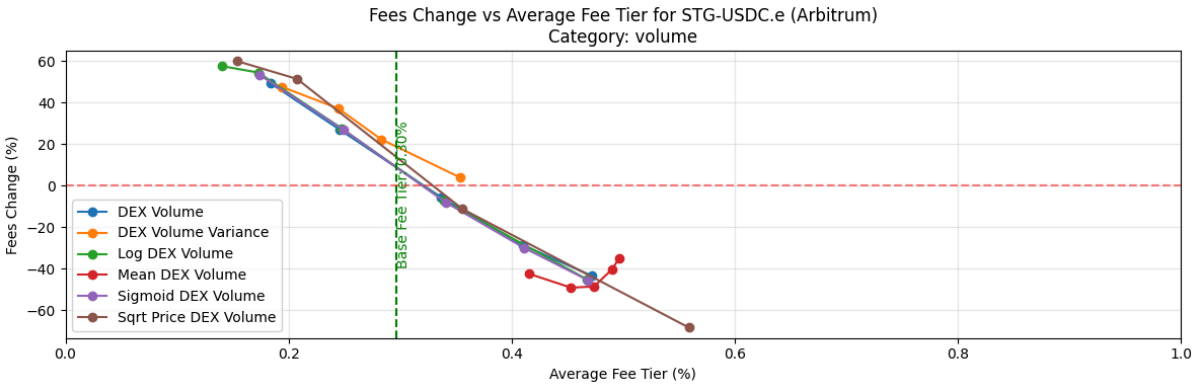

Figure 23: Change in collected fees and average fees for volume-based STG - USDC functions

Out of all studied functions, the volume-based functions can increase the profit the most. The volume-based function can achieve 60% higher fees while decreasing the average fee.

3.7. RDNT - WETH on Arbitrum

The RDNT-WETH pool on Arbitrum showed no CEX DEX arbitrage opportunities near the base fee percentage. Likely, the high liquidity and low Arbitrum block times keep the pool's price closely aligned with external markets, eliminating potential arbitrage opportunities that could be captured by adjusting the fee structure.

3.8. GHST - USDC on Polygon

Figures 24, 25, 26 and 27 illustrate different dynamic fee functions' changes in collected fees for the GHST-USDC pool based on static gas, volatility, and volume, respectively. As visualized in Figure 24 collected fees cannot be increased by lowering the average fee.

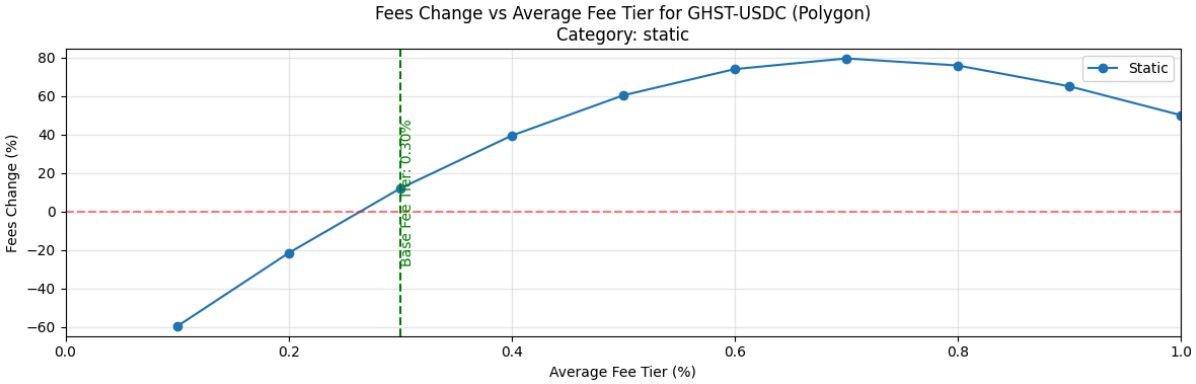

Figure 24: Static fees for GHST-USDC

As visualized in Figure 25 out of gas-based functions no function has both a higher profit and lower average fee than the current fee function. However, logarithmic and square root functions can achieve higher profit while also having a higher average fee. These functions increase the profitability by 60%.

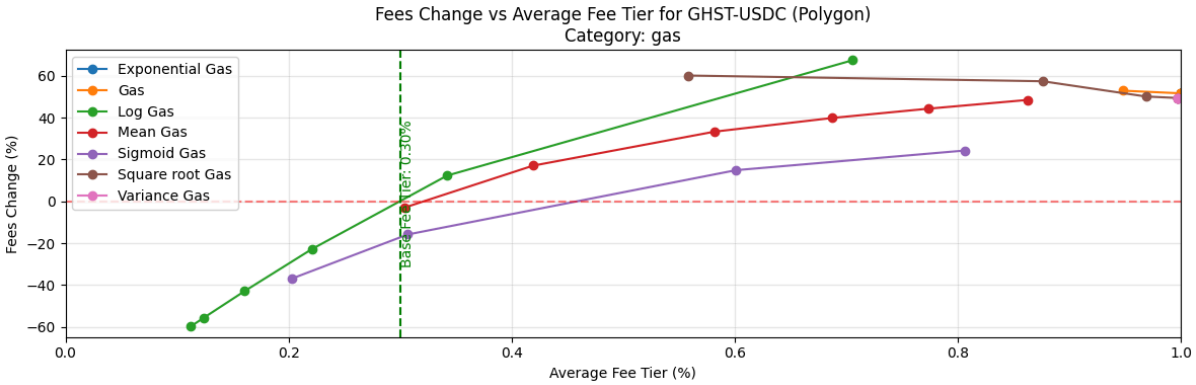

Figure 25: Change in collected fees and average fees for gas-based GHST - USDC functions

As visualized in Figure 26 no volatility-based function can increase the profits. The high variance in price volatility also

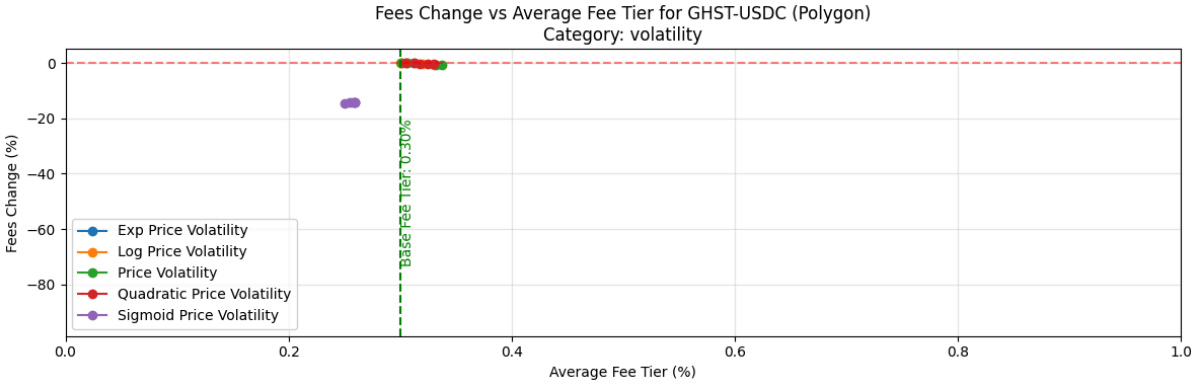

Figure 26: Change in collected fees and average fees for volatility-based GHST - USDC functions

As visualized in Figure 27, the volume-based functions perform exceptionally well compared to both base fees and the static fee function. Out of these functions multiple can achieve 40% increase in profits at the base fee percentage and 70-80% increase at the higher levels.

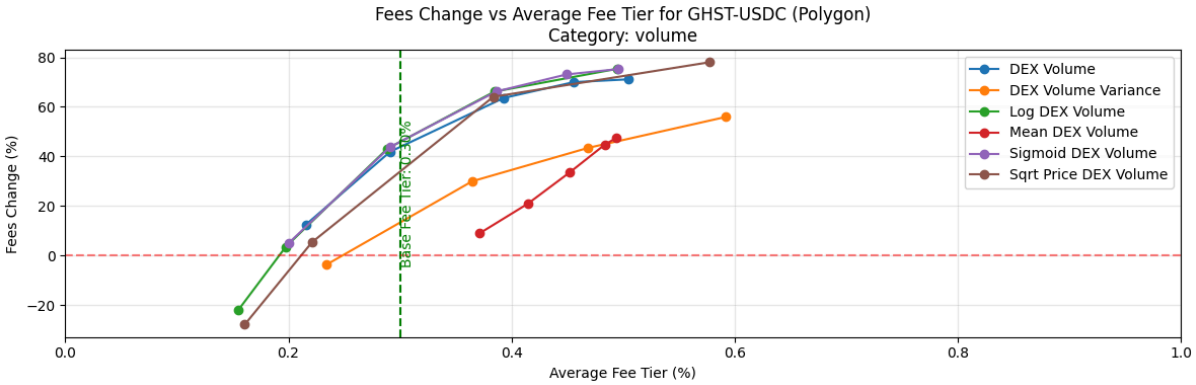

Figure 27: Change in collected fees and average fees for volume-based GHST - USDC functions

Out of all studied functions, volume-based models, particularly logarithmic and square root functions, show the most promise for the GHST-USDC pool on Polygon. These could potentially increase profits by up to 80% compared to the current static fee model, while maintaining reasonable average fees. Gas-based functions show some potential but with higher average fees, while volatility-based functions offer little improvement. Implementing a volume-based dynamic fee structure could significantly enhance the pool's performance.

4. Discussion and conclusion

Our analysis of dynamic fee functions across various pools and blockchains reveals several key insights:

1. **No Universal Solution**: Different pools demonstrate varying optimal functions and parameters for dynamic fees. Just optimising static fee levels led to significant improvements, sometimes outperforming dynamic models. This heterogeneity suggests that a one-size-fits-all approach is not feasible, and fee structures should be tailored to the specific characteristics of each pool.

2. **The potential of DEX Volume-Based Fee Functions**: Despite there not being a universal solution, fee functions based on DEX trading volume showed significant potential across most pools. This indicates that incorporating volume data into fee calculations could be a promising direction for improving pool performance.

3. **Optimal Scenarios for Dynamic Fees**: The most substantial potential benefits from dynamic fee functions were observed in scenarios where the total fees collected were higher than the base fee and the static fee while maintaining a similar or lower average fee percentage. This suggests that dynamic fees can capture more value from arbitrage opportunities without necessarily increasing the overall cost for traders.

4. **Impact of Block Times**: Pools on networks with shorter block times showed less opportunity for arbitrage and potential for improvement through dynamic fees compared to Ethereum mainnet. This is likely due to the decreased time for prices to change between blocks leading to fewer arbitrage opportunities in faster networks.

5. **Gas Price Sensitivity**: For some pools, particularly on the Ethereum mainnet, gas price-based fee functions showed promising results. This highlights the importance of considering network congestion and transaction costs in fee calculations.

6. **Volatility Considerations**: Volatility-based fee functions demonstrated varying degrees of success across different pools. While effective for some pairs, they were not universally beneficial, underscoring the need for careful evaluation of market conditions when implementing such models.

In conclusion, our study demonstrates that dynamic fee functions have the potential to improve the performance of AMM pools, particularly in capturing more value for LPs from arbitrage opportunities. However, the effectiveness of these functions varies significantly depending on the specific characteristics of each pool, the underlying blockchain network, and market conditions.

Further research could explore the long-term effects of dynamic fees on liquidity provision and trading behavior.

Appendix

Example DEX volume fee function implementation

```
interface IWeightedPool {
    function getNormalizedWeights() external view returns (uint256[] memory);
}

interface IVault {
    function getPoolTokens(bytes32 poolId) external view
        returns (address[] memory tokens, uint256[] memory balances, uint256 lastChangeBlock);
}

contract DynamicSwapFeeCalculator {
    using SafeMath for uint256;

    // Constants
    uint256 public constant WEIGHT_PRECISION = 1e18;
    uint256 public constant FEE_PRECISION = 1e18;

    // State variables
    uint256 public baseFeeBps;
    uint256 public maxFeeBps;
    IVault public vault;

    constructor(uint256 _baseFeeBps, uint256 _maxFeeBps, address _vaultAddress) {
        require(_baseFeeBps < _maxFeeBps, "Base fee must be less than max fee");
        require(_maxFeeBps <= 10000, "Max fee cannot exceed 100%");
        baseFeeBps = _baseFeeBps;
        maxFeeBps = _maxFeeBps;
        vault = IVault(_vaultAddress);
    }

    function calculateSwapFee(
        bytes32 poolId,
        address poolAddress,
        uint256 swapAmount,
        uint256 tokenIndex
    ) public view returns (uint256) {
        IWeightedPool pool = IWeightedPool(poolAddress);
        uint256[] memory weights = pool.getNormalizedWeights();
        require(weights.length == 2, "Only supports 2-token pools");

        (address[] memory tokens, uint256[] memory balances, ) = vault.getPoolTokens(poolId);
        require(tokens.length == 2, "Only supports 2-token pools");
        require(tokenIndex < 2, "Invalid token index");

        // Normalize the reserve of the token being swapped to 50/50
        uint256 normalizedReserve =
balances[tokenIndex].mul(WEIGHT_PRECISION.div(2)).div(weights[tokenIndex]);

        // Calculate the swap ratio
        uint256 swapRatio = swapAmount.mul(FEE_PRECISION).div(normalizedReserve);

        // Calculate the dynamic fee
        uint256 dynamicFeeBps = baseFeeBps.add(
            swapRatio.mul(maxFeeBps.sub(baseFeeBps)).div(FEE_PRECISION)
        );

        // Ensure the fee doesn't exceed the maximum
        return dynamicFeeBps > maxFeeBps ? maxFeeBps : dynamicFeeBps;
    }

    function calculateFeeAmount(uint256 swapAmount, uint256 feeBps) public pure returns (uint256) {
        return swapAmount.mul(feeBps).div(10000);
    }
}
```