# Scalable and Gas Optimized Arbitrage BOTS for Decentralized Exchanges in High Frequency Trading

A. Dhelpra [1] and G. Soldar [2]
University of Brighton, School of Architecture Technology and Engineering, Brighton, UK

Tel.: + 441273600900, fax: + 441273642405
[1]E-mail: a.dhelpra1@uni.brighton.ac.uk
[2]Email: g.soldar@brighton.ac.uk

***Summary:*** ***This paper addresses the absence of automated trading in the context of Decentralised Finance (DeFi) platforms and presents a novel and peer-to-peer approach to cryptocurrency exchanges. Decentralised Exchanges (DEXs) have been introduced as an innovative way to address the shortcomings of traditional financial trading, which is characterised by its centralised and cumbersome nature. While automated trading of crypto assets has gained widespread adoption, the emergence of such practices within the domain of DeFi has yet to be fully realised. The present study seeks to mitigate scalability and optimisation issues that are typically encountered by cryptocurrency arbitrage bots, which can compromise their profitability. Ultimately, this research is intended to inform best practices in automated trading for DeFi, and to facilitate the advancement of more effective, reliable, and secure approaches to cryptocurrency high frequency trading.***

**Keywords—*High-Frequency Trading, Crypto Arbitrage Trading, Solidity, Gas Optimization, Decentralized Finance, Decentralized Exchanges***

## I. Introduction

The core ideas behind blockchain technology emerged in the late 1980s and early 1990s when Leslie Lamport developed the Paxos protocol, and in 1990 submitted the paper The PartTime Parliament to ACM Transactions on Computer Systems. The paper described a consensus mechanism to achieve agreement on a result between computers in a network where the computers or the network itself may be unreliable [1].

Since then, blockchain and all the technologies that derive from it, such as crypto assets, have started to gain more and more popularity, bringing new ideas on the table such as decentralisation, transparency, anonymity and security. A key aspect of blockchain technology is that users can store their tokens on a wallet address, where their holdings are cryptographically held safe. Access to the wallet is granted through a private key, which only the user has control over. Wallets can further be divided into two categories:

- Custodial wallets: these wallets essentially work like traditional brokerage accounts or stock exchanges, meaning that their private key is managed by a third party.
- Non-custodial wallets: The private key of these wallets is managed entirely by their owners. The tradeoff is that users can't recover their funds in case they lose their private key, but it allows them to preserve anonymity and censorship-resistance.

Based on their characteristics, custodial wallets form the foundational pillar of the so-called Centralised Finance or CeFi, which represents a traditional centralised financial structure applied to the cryptocurrency industry. Their inherent attributes make them a crucial component of CeFi.

On the contrary, Decentralised Finance (DeFi) is an emerging field that leverages blockchain technology to establish a fresh financial system, distinct from CeFi. DeFi aims to eliminate the need for intermediaries such as banks and brokers, providing users with the ability to trade assets in a permissionless manner. This is made possible through decentralised exchanges (DEXs), which serve as a vital component of the DeFi ecosystem [2].

Decentralised exchanges can sometimes show price differences for the same assets, which can be exploited to make profits by buying an asset at a lower price in one market and reselling it at a higher price in another market. In economic and financial terms, this operation is called arbitrage [3]. Since these kinds of opportunities usually last only a matter of a few seconds, before the discrepancy is converged by another trader, they require the operation to happen quicker than the competition. For this reason, electronic trading is considered by both single traders and financial institutions as a very efficient trading method for arbitraging, and even more so when the computer program is able to transact a large number of orders in fractions of a second, a method called High-Frequency Trading or HFT [4].

## II. Gas Consumption

When users send a transaction to the Ethereum network, they set a gas limit, which is the maximum amount of gas units they are willing to pay for the transaction. The gas fees are therefore the product of the gas limit and the gas price, which is the amount of Gwei that a user is willing to pay for each unit of gas consumed in a transaction.

$$\textbf{\textit{Gas fees}} = \textit{gas limit} * \textit{gas price}$$

Gas fees are essential for the maintenance of the network, as they are rewarded to validators for using computational power to approve and add the block to the blockchain. This means that validators are more willing to approve a transaction with a higher gas price, as this means a higher reward for them; hence, setting a higher gas price will result in a transaction getting approved faster.

## III. DECENTRALIZED ARBITRAGE BOTS

The system presented in this paper draws inspiration from James Bachini's endeavour to develop a DEX arbitrage bot [5], and it focuses on enhancing scalability and gas optimization to overcome the limitations of traditional, centralized, and inefficient financial trading frameworks.

The main component of this solution, shown in Figure 1, is the 'Arbitrage' smart contract, which can be deployed on a given network and can fetch pair prices from various exchanges on that blockchain, verify the contract's balance, and execute transactions. The system also utilises external smart contracts and libraries to calculate profits, trade, and deposit Wrapped Ethereum (WETH) to the bot while ensuring the security of the smart contract.

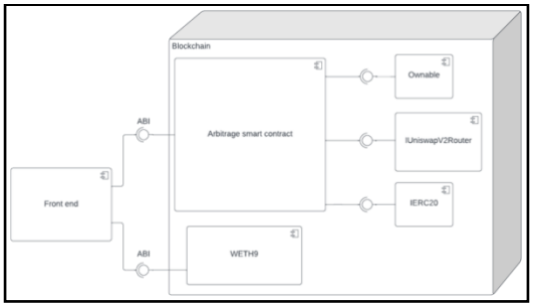


Figure 1. Component diagram showing sub-components of the backend

Communication between the frontend and backend is facilitated through the Application Binary Interface (ABI) of the Arbitrage and WETH9 smart contracts. The ABI provides information about function names, input/output parameters, and data types used by the contract, enabling the frontend to call functions and read data.

The bot can be adjusted to scale the number of DEXs and tokens to track by modifying the relevant addresses in the configuration file, as well as the network where to deploy the bot to. This provides flexibility on the markets that the user decides to track, which can be essential in order to mitigate high gas fees and slippage.

Indeed, the program can perform its transactions on an emerging network that represents a good balance between an ecosystem that has not been targeted by other trading bots yet and a liquid ecosystem that allows only a small margin between the quote price and the actual price of an asset for large transactions. This ensures that arbitrage opportunities are not taken from other bots and reduces the risk of paying gas fees for an incomplete transaction, while also reducing slippage when scaling up the amounts traded. It is also worth noting that some ecosystems not only possess the aforementioned advantages, but also have the added benefit of offering extremely low transaction fees, significantly reducing the risk of high gas fees.

Figure 2 shows the series of interactions involved when the arbitrage bot fetches price data from DEXs to detect arbitrage opportunities. It operates by first retrieving the equivalent amount of ETH for a specific token, such as USDC, from the initial decentralised exchange (DEX). It then proceeds to inquire about the quantity of initial tokens that can be acquired from the second DEX using the ETH obtained from the first DEX.

If the estimated returned amount is higher than the current balance, meaning a profit exists, the trades get implemented. If this does not happen, the bot will recursively repeat this cycle across a list of tokens and DEXs until an opportunity is found.

During the execution of trades, a similar sequence of interactions occurs, but this time the focus shifts from fetching prices to actually swapping tokens. However, there is an important requirement to be met: the final balance must be higher than the initial balance. If this condition is not fulfilled, the trades are reverted. Such a property ensures that the program does not perform an arbitrage operation at a loss. It is however important to notice that any paid gas fees are not recovered, which may result in a loss nevertheless.

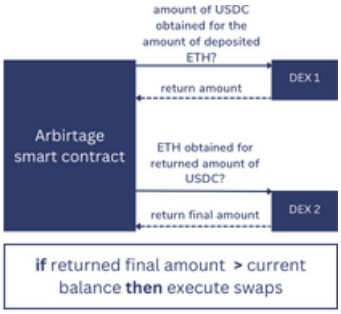


Figure 2. A visual representation of the interactions of the smart contract with other DEXs

## IV. IMPLEMENTATION

Gas consumption has been optimised by implementing specific design patterns that address common gas-related issues. These design patterns are discussed in the article Design Patterns for Gas Optimization in Ethereum [6]. In addition to these, the Solidity optimiser, developed by the Ethereum Foundation, has been used to optimise smart contract bytecode by minimising gas consumption and increasing performance. It achieves this by performing a series of code transformations, such as removing redundant code and replacing expensive operations with cheaper ones, resulting in smaller and more efficient bytecode. It is possible to activate the gas optimiser using the "--optimizer" parameter on the Solidity compiler, or using the 'pragma solidity' statement at the start of the smart contract.

### *A. WETH9 Smart Contract*

The frontend communicates with the WETH9 and Arbitrage smart contracts using the Web3.js library. The deposit function of this application calls two methods from the WETH9 contract: it first approves the WETH9 smart contract to withdraw the inserted amount of funds from the owner's account using the contract's approve function, and then transfers the same amount from the owner's address to the bot's contract address by calling the transferFrom function. Both of these methods modify the state of the blockchain, which means that they require gas to be paid in order to complete them. The gas limit set for completing transactions is of 200,000 gwei, which corresponds to 0.0002 ETH.

### *B. IUniswapV2Router Smart Contract*

Moreover, the Arbitrage contract is dependent on the IUniswapV2Router contract to estimate the profit of trades and execute them. This contract is part of the Uniswap

decentralised exchange protocol and it is responsible for managing the liquidity pools used by the DEX, as well as facilitating the addition and removal of liquidity from those pools. In addition to that, when a user submits a trade in the Uniswap DEX, this contract determines the most efficient path for the trade to take before delegating its execution to the UniswapV2Pair contract (Uniswap documentation contributors, n.d.). The majority of decentralised exchanges in the DeFi domain are a fork of the Uniswap DEX, meaning that the functionality of the UniswapV2Router can be used for other DEXs too. This makes the bot really flexible on the choice of DEXs to track.

To simulate an arbitrage operation, the Arbitrage contract estimates the returned amount when swapping the current balance of a given token A for token B on a first DEX, and then it does the same for the returned amount of token B to be swapped back with token A on a second DEX. To achieve this, it uses the IUniswapV2Router's function getAmountsOut(), which retrieves the price of a given pair in the specified decentralised exchange. A visual representation of this process is shown in Figure 3.

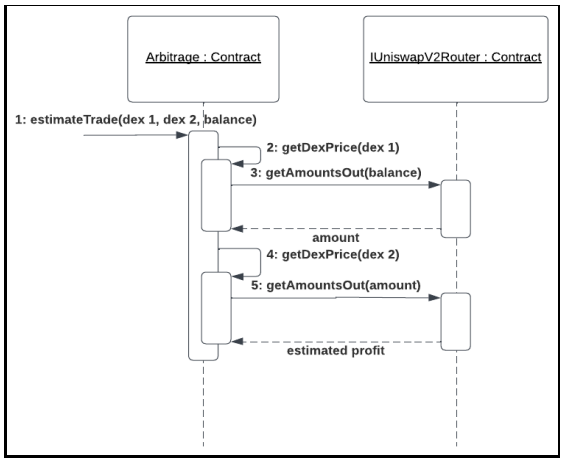


Figure 3. Sequence diagram of potential profit estimation

### C. IERC20 Smart Contract

Furthermore, the Arbitrage contract is dependent on the IERC20 smart contract, which is an interface of the ERC20 standard as defined in the Ethereum Improvement Proposals (Vogelsteller and Buterin, 2015). It includes the functions required to approve and execute transactions discussed earlier for the WETH9 contract, which itself is an implementation of this standard. It uses its approve() function to approve the two swaps occurring in the two different DEXs. The tokens are transacted using the IUniswapV2Router's function swapExactTokensForTokens(), first on an exchange, then the other. The Arbitrage contract requires the starting balance to be higher than the balance after the trades are completed. This is an important feature that allows to revert the transaction if the estimated profits did not match the actual results, even after the transactions have been committed but yet validated on the blockchain, ensuring fund security. A sequence diagram showcasing this process is shown in Figure 4.

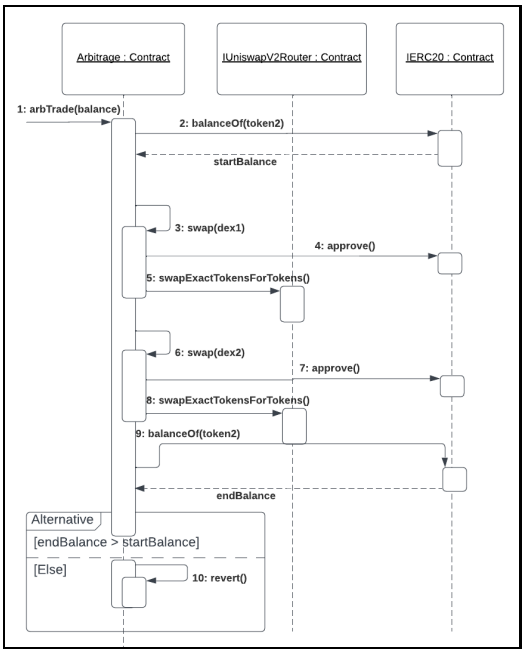


Figure 4. Sequence diagram of token swaps

### D. OpenZeppelin Libraries

In addition to the aforementioned libraries and frameworks, Openzeppelin battle-tested libraries of Ethereum smart contracts have been used to implement token standards to minimise risk and provide better security. The arbTrade function implements the nonReentrant modifier provided by the Reentrancy contract developed by OpenZeppelin, which has checks in place to prevent reentrancy attacks, as well as the onlyOwner modifier. This is a modifier provided by the Ownable contract also developed by OpenZeppelin, that ensures that this function can only be called by the owner of the contract.

## V. RESULTS

### A. Gas testing

Figure 5 displays a report where the methods recoverEth and recoverTokens from the Arbitrage contract are evaluated by the hardhat-gas-reporter tool. The results show an average gwei cost of 30421 for the method recoverEth and 36806 for the method recoverTokens. The price per gas limit, 24 gwei, is obtained by the gas reporter from the Eth Gas Station. The report also reveals the cost converted to USD, which is based on the USD/ETH pair of approximately 1569 at the time of measurement. These results are averaged over 200 runs.

Furthermore, the reporter presents the cost of deploying the contract to the Ethereum network, which is 4.3% of the block limit of 30000000 gas. This corresponds to a cost of 48.35 USD.

| Solc version: 0.8.16 | Optimizer enabled: false | Runs: 200 | Block limit: 30000000 gas |
|---|---|---|---|
| Methods | 24 gwei/gas | | 1569.03 usd/eth |
| Contract · Method | Min | Max | Avg | # calls | usd (avg) |
| Arbitrage · recoverEth | - | - | 30421 | 1 | 1.15 |
| Arbitrage · recoverTokens | - | - | 36806 | 1 | 1.39 |
| Deployments | | | | % of limit | |
| Arbitrage | - | - | 1283864 | 4.3 % | 48.35 |

Figure 5- A report generated by the plug-in hardhat-gas-reporter when the solidity optimizer is not enabled.

After enabling the Solidity optimiser, the tests were re-evaluated. The results indicate that the average gwei cost for the recoverEth method was 30306, and for recoverTokens, it was 35854. These figures suggest an improved usage of gas of 0.4% for the recoverEth method and 2.59% for recoverTokens(). Additionally, the deployment of the contract decreased to 2.7% of the gas limit, revealing a significant improvement of almost 37% from the previous test. The results are displayed in Figure 6.

| Solc version: 0.8.16 | Optimizer enabled: true | Runs: 200 | Block limit: 30000000 gas |
|---|---|---|---|
| Methods | 23 gwei/gas | | 1569.04 usd/eth |
| Contract · Method | Min | Max | Avg | # calls | usd (avg) |
| Arbitrage · recoverEth | - | - | 30306 | 1 | 1.09 |
| Arbitrage · recoverTokens | - | - | 35854 | 1 | 1.29 |
| Deployments | | | | % of limit | |
| Arbitrage | - | - | 800683 | 2.7 % | 28.89 |

Figure 6 - A report generated by the plug-in hardhat-gas-reporter when the solidity optimizer is enabled.

### *B. Backtesting*

The bot was run for 17 consecutive days on the Goerli test network from the 9th March 2023 to 24th March 2023 on a AWS-hosted EC2 Instance.

The backtesting results revealed a surprising profit of 229.42% during the analysed period, going from a starting balance of 0.05 WETH to a final balance of 0.164711147 WETH. It was noted that the number of completed transactions was not linear; in fact, 10 out of 17 days of testing showed no transactions completed. The day with the largest profit was day 6, where the balance bot increased by 0.050183808 WETH, which corresponds to a 53.5% increase from the previous day.

The results also showed that the bot did not incur any losses, indicating a trading strategy with minimal risk. This is visible in Figure 7.

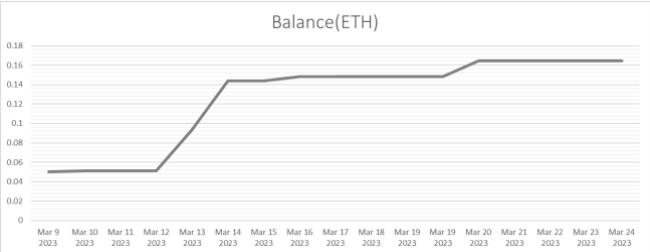


Figure 7 – Bot profits from 09/03/2023 to 24/03/2023

## VI. CONCLUSIONS

The testing performed showed that the bot is able to detect arbitrage opportunities and take advantage of those in the Goerli test network with multiple tokens against the WETH token. It also showed that the gas consumption is minimised and that best practices such as limiting the use of storage or unpacked integers have been followed. This is confirmed by a surprising performance in the testnet, which saw an increase of funds from 0.5 to more than 0.16 WETH, which corresponds to a 220% profit.

Yet, the research could have been enriched by considering other data. For example, this project does not take into consideration the time spent for each transaction to complete, which can be an important factor in determining the efficiency of the algorithm and the appropriate gas limit. In addition to this, some patterns could be revealed by analysing the missed arbitrage opportunities by the bot, to measure the efficiency of the detection algorithm. Moreover, it would have been beneficial to gather data on the performance of the bot considering different exchanges and tokens. It is also worth noting that the performance of the bot could significantly change on a mainnet, as arbitrage bots are known to be susceptible to frontrunning attacks.

In addition, the arbitrage bot's strategy could be enhanced by leveraging the tools that DeFi has to offer. One of these are flash loans (Wang et al., 2021). These are loans that users can borrow without any collateral if the borrowed assets are paid back within the same blockchain transaction.

## VIII. REFERENCES


[1]. Lamport, L. (1998). The part-time parliament. ACM Transactions on Computer Systems, [online] 16(2), pp.133–169.:https://doi.org/10.1145/279227.279229

[2]. Qin, K., Zhou, L., Afonin, Y., Lazzaretti, L. and Gervais, A. (2021). CeFi vs. DeFi -- Comparing Centralized to Decentralized Finance. arXiv:2106.08157 [cs, q-fin]. [online] Available at: https://arxiv.org/abs/2106.08157

[3]. Dybvig, P.H. and Ross, S.A. (1989). Arbitrage. Finance, pp.57–71. doi:https://doi.org/10.1007/978-1-349-20213-3_4

[4]. Aldridge, I. (2013). High-Frequency Trading: A Practical Guide to Algorithmic Strategies and Trading Systems, John Wiley & Sons.

[5]. Bachini, J. (2022). Introduction To DEX Arbitrage, Available at: https://jamesbachini.com/dex-arbitrage

[6]. Marchesi, L., Marchesi, M., Destefanis, G., Barabino, G. and Tigano, D. (2020). Design Patterns for Gas Optimization in Ethereum. 2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE).