Detecting Cryptocurrency Wash Trading

Jannis Kappel
TU Berlin
Berlin, Germany

Alexander Schwind
TU Berlin
Berlin, Germany

Paul Vidal
TU Berlin
Berlin, Germany

Abstract

This project focuses on reproducing and optimizing a wash trade detection algorithm for decentralized exchanges (DEXs), with a particular focus on IDEX transaction data. We re-implemented the original algorithm in Python using more efficient libraries such as networkx and introduced parallelization across independent computational steps. These changes resulted in a 5.5× overall speedup, reducing total runtime from 45 minutes to 9 minutes. We further extended the algorithm with a sliding window approach, allowing detection of manipulation patterns spread across overlapping time intervals. This enhancement improved detection coverage, increasing the number of identified wash trades from 191,967 to 217,234. Additionally, we incorporated a newly collected dataset spanning 2020 to 2025, which revealed only a modest increase in wash trading, suggesting a decline in such activity on IDEX in recent years. While the analysis focused solely on IDEX due to data format differences with other DEXs like EtherDelta, the methods developed are generalizable and offer a scalable foundation for real-time or cross-platform wash trading detection.

Keywords

Wash Trading, Decentralized Exchanges, Blockchain

1 Introduction

Wash trading is a manipulative trading practice in which the same entity (or colluding entities) rapidly buys and sells an asset to create a false impression of market activity. By generating illusory trading volume without genuinely changing market positions or incurring real market risk, wash traders aim to deceive other participants regarding an asset's demand and liquidity. This practice is illegal in most regulated markets and is explicitly defined by authorities such as the U.S. Commodity Futures Trading Commission (CFTC) as trading that "gives the appearance that purchases and sales have been made, without incurring market risk or changing the trader's market position." In the cryptocurrency domain, wash trading misleads investors by inflating volume metrics, influences asset rankings on exchanges, and generally undermines trust in market integrity. Detecting and curbing such manipulation is therefore a critical challenge for the crypto industry.

The rise of decentralized exchanges (DEXs) has introduced both new opportunities and challenges for wash trading detection. Unlike centralized exchanges, where off-chain trade records and opaque data reporting make verification difficult, DEX platforms record every trade on a public blockchain ledger. This transparency enables researchers to directly analyze account-level trading patterns rather than relying on exchange-reported data. However, despite the costliness of on-chain trades (due to transaction fees), suspicious volume spikes and self-dealing behaviors persist, indicating the presence of wash trading on decentralized platforms. Identifying such manipulative trading on DEXs, therefore, requires sophisticated algorithmic approaches and robust analysis of on-chain activity.

1.1 Summary of the examined paper

A pivotal contribution to the study of wash trading on decentralized exchanges was made by Friedhelm Victor and Andrea Marie Weintraud in their 2021 paper, "Detecting and Quantifying Wash Trading on Decentralized Cryptocurrency Exchanges" [1]. The authors focused on Ethereum-based DEXs utilizing a limit-order-book model—specifically, EtherDelta and IDEX—to systematically identify and quantify wash trading activities.

The methodology employed in the paper is grounded in the use of on-chain data and advanced graph algorithms. The authors modeled the network of trades as a directed graph, with addresses as nodes and trades as edges. They then applied strongly connected component (SCC) analysis to detect groups of accounts engaged in cyclic trading, which are indicative of potential wash trading. Each SCC, representing a set of addresses with bidirectional trading relationships, was further analyzed to identify instances where trade volumes netted out to nearly zero, indicating that traders ended up with effectively unchanged positions—a key legal criterion for wash trading.

The results revealed pervasive wash trading on the analyzed DEXs. Over 159 million USD worth of cryptocurrency transactions were attributed to wash trades on EtherDelta and IDEX. The majority of these activities consisted of self-trades or ping-pong trades between two accounts, although more complex multi-account cycles were also observed. The analysis showed that over 30% of tokens listed on these exchanges exhibited signs of wash trading, with some tokens being almost exclusively wash traded. These findings demonstrate that a significant fraction of reported volume on decentralized exchanges can be attributed to manipulative trading activity, underscoring the need for effective detection and mitigation mechanisms. The authors also highlighted the necessity to adapt these detection methods to newer DEX models, such as automated market makers (AMMs), which now dominate the decentralized trading landscape.

1.2 State of EtherDelta and IDEX

EtherDelta, once a pioneering on-chain orderbook exchange, ceased front end operations following technical issues and regulatory action. Its smart contract persists on the Ethereum blockchain and still reflects an impressive number of transactions historically. However, user engagement and liquidity have effectively evaporated. In the post 2018 period, only sporadic deposits, withdrawals, or archival trades occur. Consequently, recent data extracted from EtherDelta's contract will be extremely sparse and unlikely to support robust statistical analysis. IDEX, by contrast, adopted a hybrid model and continued enhancing its offering. Notably, in early 2025 IDEX rebranded to Kuma, migrated its perpetuals platform to Berachain, and expanded support for omnichain deposits across several Layer 2 networks Messari. Despite some recent regulatory pressure, reflected in token price volatility, the protocol remains operational and maintains user activity. The methodology employed in the referenced paper relies on tracing discrete buyer-seller interactions within smart contract event logs. In EtherDelta's dormant state, applying such methods to recent data would yield negligible results due to near zero trade volume. Researchers should expect extremely limited bilateral trades post shutdown. For IDEX/Kuma, however, continued operation means new transaction data—though the platform's structural evolution matters. If trades still occur under the legacy hybrid order book model, the standard buyer seller evaluation remains valid. But after the migration to Kuma/Berachain, a significant share of transactions may involve perpetual swaps or cross chain pools, reducing transparency of counterparty pairs and thereby the availability of data with parameters required for the referenced method.

1.3 Goal and Methodology

Building on the foundation established by Victor and Weintraud the goal of this project is to study and improve the presented wash trade detection method. By gathering more up-to-date on-chain data and adapting and improving the algorithm, we seek to enhance the efficiency and coverage of wash trading detection in decentralized markets. The overarching objective is to contribute a refined methodology that not only reproduces the key findings of prior work but also provides results on the latest data in a more efficient manner. Such improvements can help regulators, exchanges, and the community to better identify illicit trading behaviour and uphold fair trading practices as the cryptocurrency industry continues to mature. This paper will propose possible adaptations for the algorithm and introduce a dataset update strategy and evaluate the performance and results achievable with the improvements and compare them with the baseline.

1.4 Related Work

Wash trading in cryptocurrency markets has been widely documented, both on centralized and decentralized platforms. On many unregulated centralized exchanges, a significant portion of reported volume is believed to be artificial. For example, a 2019 analysis by Bitwise Asset Management revealed that around 95% of the Bitcoin trading volume reported on CoinMarketCap was fake, generated by exchanges through non-economic wash trades. This practice is often motivated by exchange listing incentives or to create a misleading appearance of liquidity. The lack of transparency in centralized exchanges makes such manipulations hard to verify; researchers have resorted to indirect clues like statistical irregularities in trade data. Common indicators include unnaturally uniform trade sizes or suspicious time clustering, and techniques based on Benford's Law or trade size histograms have been proposed to flag fake volume. Bitwise's report to the U.S. SEC in 2019 underscored the severity of the issue, noting that only a handful of major exchanges accounted for nearly all real trading activity.

Academic studies have further explored wash trading detection methods. Victor and Weintraud (2021) were among the first to rigorously analyze wash trading on decentralized exchanges using on-chain data. They observed that fully anonymous, blockchain-based DEXs are particularly vulnerable to wash trading, finding that over 30% of tokens on certain DEXs showed signs of wash trades. Their approach leveraged graph algorithms to identify cyclical trading patterns. In parallel, other researchers have investigated alternative detection approaches. For instance, Le Pennec et al. proposed machine learning models to classify wash trading behaviour across different blockchain networks, while Sila et al. (2024) examined macro-level factors and found that market volatility and exchange inflows correlate strongly with wash trading prevalence. These complementary approaches – from statistical fingerprints to ML classification – highlight the multifaceted nature of wash trade detection.

2 Concept

In this project, the primary responsibility was to re-implement and optimize the detection algorithm proposed in the original paper, with the goal of significantly improving runtime performance. We investigated the algorithm and found multiple opportunities to improve the algorithm.

2.1 Rolling Mean Calculation

The volume matching algorithm relies on computing aggregate statistics like the total traded volume per participant and the mean trade size within a window. In the original formulation, for each candidate set of trades the algorithm may recalculate these metrics repeatedly. Recomputing sums and means from scratch is computationally inefficient, especially when analysing overlapping windows or iteratively removing trades to find a wash trade subset. There is a clear opportunity for optimization by reusing partial computations. The motivation is to avoid redundant calculations of volume totals or averages whenever the data changes only slightly. We introduce rolling computations for the volume matching algorithm, using incremental updates instead of full re-computation. In practice, this means maintaining running totals and counts of trade volumes that can be quickly adjusted as the window shifts or as trades are added/removed, rather than recalculating everything. Within the volume matching algorithm, when iteratively removing the last trade to test a smaller subset, we decrement the cumulative sums of each participant's buy and sell volumes without recomputing the entire sum from scratch. By caching these intermediate sums and counts in memory, the algorithm can compute the new mean trade size or updated trader positions with minimal work. In algorithmic terms, this technique turns expensive re-computation into cheap incremental updates – effectively amortizing the cost over many operations, scaling in efficiency with the size of the dataset on which the mean operation is applied. The main trade-off for this optimization is increased memory usage: the algorithm must store caches of running totals and data structures to enable quick updates. This represents a classic time–memory trade-off – we expend additional memory to save computation.

2.2 Parallelization

The volume matching step of the detection algorithm must evaluate many candidate trade sets for wash trading. Even after narrowing down candidates via strongly connected components, the algorithm still needs to analyse numerous trades within each time window for each candidate. As blockchain usage grows, the number of trades and potential wash trading instances increases dramatically, leading to heavy computational demand. Running the volume matching sequentially can become a bottleneck, therefore, optimizing runtime is essential to scale the method to larger datasets and to potentially enable near real-time analysis of streaming trade data.

Algorithm 1: VolumeMatchingParallel: Distinct Time-windowed wash trade detection

```
Data: Trades, relevant SCCs, SCC-trader map
Result: Detected Wash Trades
1  Initialize list of window sizes (1h, 1d, 1w)
2  Set all trades' wash labels to False
3  foreach scc_id in relevant SCC hashes do
4      Get scc_traders from global map
5      Filter scc_trades where both buyer and seller are in
         scc_traders and not labeled as wash
6      foreach window_size in window sizes do
7          Create timestamp windows breaks
8          Assign distinct window labels to trades via pd.cut
9          Group trades by (token, window)
10         Run DetectLabelWashTrades in parallel on each
             group
11         Collect flagged transaction hashes into all_hashes
12         Label those transactions in trades as wash trades
13     end
14 end
15 return updated trades, wash_trades
```

We introduce a parallelized matching algorithm that distributes the volume matching computations across multiple processors or threads. In our design, independent tasks – such as checking different SCC-derived candidate groups or separate time windows – are executed concurrently. By partitioning the workload, the algorithm exploits modern multi-core architectures to reduce overall processing time. This parallelization does not change the underlying logic of volume matching; instead, it orchestrates multiple matching operations to run in parallel rather than in sequence.

Parallel execution yields a significant reduction in overall computation time, roughly proportional to the number of parallel workers up to the limits of coordination overhead and hardware scaling. This improvement makes the detection algorithm far more scalable. For instance, if the dataset doubles in size, a parallel algorithm can handle the increase by utilizing additional cores, whereas a single-threaded approach would see a roughly two-fold increase in runtime. By reducing computational latency, the parallelized approach enables analysis of much larger datasets or more frequent re-runs of the detection process. This is particularly important given the continual growth of on-chain trading activity and the need for timely detection of manipulative patterns.

Algorithm 2: DetectLabelWashTrades: Volume Matching Logic

```
Data: Buyers, sellers, trade amounts, number of IDs, margin
Result: Flags indicating wash trades
1  foreach trade i do
2      Update balance_map with amount bought/sold
3  end
4  for $idx = len - 1$ down to 1 do
5      Compute mean trade volume for trades 0 to $idx$
6      if mean == 0 then
7          break
8      end
9      Normalize each trader's balance by mean
10     if all balances $\leq$ margin then
11         Flag trades 0 to $idx - 1$ as wash trades
12         return
13     end
14     Roll back trade $idx$ from balance_map
15 end
```

2.3 Overlapping Time Windows

To improve the temporal resolution of wash trade detection and capture patterns that span broader or misaligned intervals, we extended the algorithm to support overlapping time windows using a sliding window approach. In the original implementation, non-overlapping time windows were used, which limited detection to trades occurring strictly within a fixed range. As a result, manipulative patterns that unfolded across window boundaries or over longer durations could go undetected.

In the updated version, for each SCC and selected time window size, we iterate over the trade timeline with a configurable stride, allowing each window to overlap with its predecessor. For example, using a window size of 10 minutes and a stride of 5 minutes, we analyze the data in partially overlapping windows, ensuring finer-grained temporal coverage.

This approach replaces the previous use of pandas.cut, which was limited to non-overlapping binning. Instead, we filter trades for each time window explicitly by timestamp, enabling full control over overlapping intervals. Each window is then processed in parallel using the same volume matching algorithm, ensuring scalability despite the increased number of windows.

2.4 Data Extension

We also want to extend the dataset by adding the trades after 05/04/2020 (frame limit of the article data) of the two smart contracts. This way we can apply our improved algorithm on new data and explore if the wash trading problem still is significant.

The idea is to retrieve transactions from the smart contracts with Etherscan, a database connected to the Ethereum blockchain. From this transaction, we would then filter and format the trades in order to have CSV files similar to the ones from the article.

Algorithm 3: OverlappingWindowExtraction: Generate Sliding Windows

```
Data: Trades, window sizes, global time range
Result: List of wash trades from overlapping windows
1  Initialize empty list windowed_groups
2  foreach window_size in window sizes do
3      Set stride as 0.75× window_size
4      Generate window_start_times from window_start to
         window_end using stride
5      foreach start_time in window_start_times do
6          Set end_time = start_time + window_size
7          Select window_trades between start and end time
8          if window_trades is not empty then
9              Append (start_time, window_trades) to
                 windowed_groups
10         end
11     end
12 end
13 return List of trades in this time window
```

Algorithm 4: Parallel SCC extraction over grouped tokens

```
Data: Trades
Result: Relevant SCCs
1  Set all trade weights to 1
2  Group trades by token into sub_trades_list
3  Run ProcessSubTrades on each item in sub_trades_list in
     parallel
4  Collect results into results
5  Initialize all_results = [] and global_scc_map = {}
6  foreach (result, local_map) in results do
7      Append result to all_results
8      Merge local_map into global_scc_map
9  end
10 Count occurrences of each SCC hash
11 Filter SCCs where occurrence ≥ 100 into relevant
12 return relevant
```

3 Evaluation

Our implementation yielded significant improvements of the runtime and detection logic. We were also able to extend the dataset to the current year but did not find a significant increase in wash trading.

For this project, all experiments and algorithm evaluations were conducted exclusively on transaction data from the decentralized exchange IDEX. While EtherDelta is another well-known DEX with historical significance, its transaction data follows a different structure and formatting, requiring a separate preprocessing pipeline.

Due to time constraints and the focus on performance optimization, we prioritized consistency in the dataset and chose not to adapt the algorithm to EtherDelta's format at this stage. Extending the analysis to EtherDelta remains a valuable direction for future work, particularly to compare wash trading behavior across platforms and validate the generality of our detection method.

3.1 Parallelization

3.1.1 Efficient Graph-Based Detection. The original implementation constructed token-wise graphs and searched for strongly connected components to detect clusters of traders potentially involved in manipulative behavior. We restructured this part of the algorithm using Python and the networkx library, which offers efficient graph processing primitives. To enable parallelism, we grouped all transactions by token in advance using a single pass over the dataset with pandas.groupby, eliminating the previous bottleneck where this grouping was redundantly computed in each iteration.

Each token group was then processed in a separate process using Python's joblib package, allowing the SCC detection algorithm to run in parallel across tokens.

In addition, we optimized the SCC extraction step. After initially identifying a multigraph for each token, we converted it into a simplified directed graph by collapsing parallel edges and assigning them weights. We then iteratively extracted SCCs by decrementing edge weights, removing edges and nodes once the weight reached zero. This strategy allowed us to extract clusters proportional to the original multigraph structure without recomputing the graph from scratch.

3.1.2 Parallel Volume Matching with Scan Sharing. Following SCC detection, each SCC was analyzed for suspicious trading activity using a custom volume matching algorithm. The algorithm inspects whether each trader in an SCC maintains a roughly neutral balance—suggesting wash trading—across multiple time windows. The detection routine is implemented in C for efficiency and uses a reverse iteration strategy to check if the net balances of all traders fall within a configurable margin of the mean transaction volume.

To further improve runtime, we introduced a memory-efficient scan-sharing strategy. Previously, each SCC was re-scanned individually, incurring substantial overhead. In the new implementation, we pre-sorted the transaction dataset once and used pandas.groupby to create a cached list of SCC-specific transaction subsets. These were then processed in parallel using multiple worker processes, minimizing redundant I/O and significantly reducing data access costs.

We also apply operation reordering to gain an even bigger runtime improvement. This way we slightly change the order of execution and find that the original algorithm is very sensitive to the order of transaction processing. In theory the results should always be the same regardless of order, but in practice this is not the case. Depending on how the trades are processed the wash label gets updated in different orders which influences the future steps. This is an important factor that should be addressed in future work.

3.1.3 Results. The parallelization in the SCC algorithm yielded a dramatic performance improvement: SCC detection runtime was reduced from 12 minutes and 43 seconds to 10 seconds — a **76× speedup** (see Figure 1).

The parallelization, along with the C-level implementation of the detection logic, reduced the volume matching runtime from 32 minutes and 47 seconds to 7 minutes — a **4.8× speedup** (see Figure 2).

Algorithm 5: ProcessSubTrades: Extract SCCs from token transaction graph

```
Data: sub_trades_list
Result: List of SCCs
1  Build a multi-directed graph $G$ from trades
2  Initialize a simple directed graph $G_{simple}$
3  foreach edge $(u, v)$ in $G$ do
4      if $u \neq v$ then
5          if $(u, v)$ exists in $G_{simple}$ then
6              Increment the edge weight
7          else
8              Add edge $(u, v)$ with weight 1
9          end
10     end
11 end
12 while $G_{simple}$ has nodes do
13     Extract strongly connected components with size > 1 as SCCs
14     if SCCs is empty then
15         break
16     end
17     foreach SCC in SCCs do
18         Add SCC to result
19     end
20     foreach edge $(u, v)$ in $G_{simple}$ do
21         Decrease edge weight by 1
22         if $weight \leq 0$ then
23             Mark edge for removal
24         end
25     end
26     Remove marked edges and isolated nodes from $G_{simple}$
27 end
28 return result
```

Table 1: Deviation of results with Operation Reordering

	Wash Trades
Original Algorithm	186,413
Parallel & Reordered	191,967

Overall, our optimized pipeline reduced total runtime from 45 minutes to 9 minutes, achieving a **5.5× overall speedup** (see Figure 3). Importantly, these changes preserved the correctness of the results, with no change in detection accuracy. The algorithm now scales effectively to over 5 million transactions, making it more suitable for production use or further research, as it enable quicker iteration and thus promotes more experimentation.

3.2 Overlapping Time Windows

This extension significantly enhances the detection capability of the algorithm. Wash trades that are spread across adjacent windows, previously undetectable due to boundary cutoff, are now properly identified. It also enables the detection of longer-duration

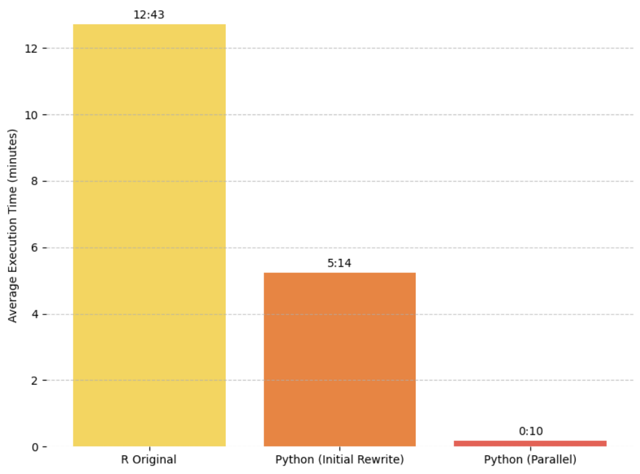

Figure 1: SCC algorithm performance comparison

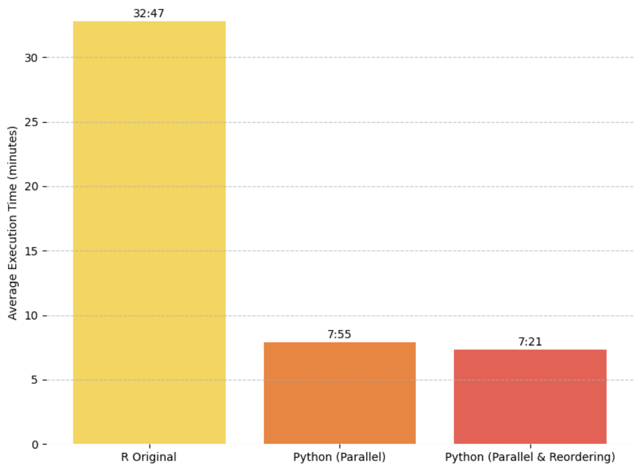

Figure 2: Volume Matching algorithm performance comparison

manipulation patterns, which are common in low-liquidity tokens where trades may be spaced further apart.

Although the introduction of overlapping windows increases the number of computations, and thereby extends the runtime of the volume matching phase to 33 minutes (up from 7 minutes), the total runtime of the full pipeline remains under the original baseline of 45 minutes (at 34 minutes). This demonstrates that the enhanced detection power is gained without regressing on overall performance goals.

The benefits of this extension are also reflected in the results: with the sliding window approach, the algorithm detected 217,234 wash trades, compared to 191,967 detected in the non-overlapping version — an increase of approximately 13%. This confirms that the improved temporal coverage enables the detection of manipulation patterns that were previously missed due to rigid window boundaries.

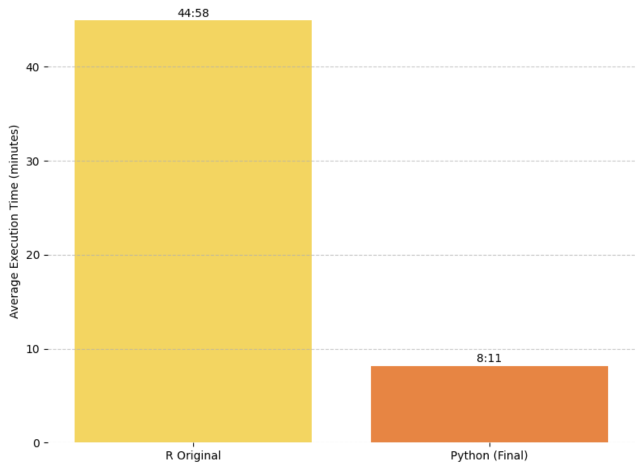

Figure 3: Full algorithm performance comparison

Table 2: Impact of Overlapping Time Windows

	Wash Trades
Original Algorithm	186,413
Overlapping Windows	217,234

3.3 Data Extension

Etherscan is an API provider so we can retrieve the transactions from an address (given in the article) with an API command launch from a Python console and do the rest of the processing there. However, it is a free version so we can only get 10,000 transactions per command. Thus, we need to make several calls and to ensure the transactions retrieved are exhaustive and distinct.
By analysing the API command, we notice that we can choose the block interval in which to search for transactions. So, by explicitly giving it different block numbers, we are sure that the transactions are distinct (because they are from different blocks). The objective is now to know how wide this interval can be so we can retrieve all the possible transactions, i.e. there are less than 10,000 transactions in the block interval.

We opted for a direct solution which is to call the database with a specific block interval : if we get 10,000 transactions, it means there are more but the API provider has limited the retrieve so we divide the interval by two and we make two API commands with the half-intervals. This way, we know that the intervals are small enough to get all the transactions from them. Then, we concatenate the obtained data to have one dataset for each smart contract.

We now need to extract only the trades, which can be identified by the *methodId*, one of their attributes. To do so, we use SQL commands on the datasets and we also only retrieve the attributes which are present in the article's datasets.

However, some information in a trade is still encoded in a hexadecimal string in the attribute named *input*. To decode it, we need the type information of the *input*, given in another attribute : *functionName*.

Algorithm 6: Transactions retrieving

```
Data: Smart contract address
Result: Table of transaction
getTx(sBlock, eBlock)
    $data$ ← getApi(address, sBlock, eBlock)
    if $data = None$ then
        $data$ ← []
    else
        if $length(data) = 10000$ then
            $m$ ← (sBlock + eBlock) / 2
            $data$ ← getTx(sBlock, $m$) + getTx($m$ + 1, eBlock)
        end
    end
    return $data$
end
return getTx(0, 99999999)
```

Finally, we obtain the database of IDEX's and EtherDelta's trades, which we can compare with the article database.

Table 3: Comparison of IDEX's and EtherDelta's trade number

	IDEX	EtherDelta
Original database	5,340,537	3,573,512
Extended database	5,887,059	4,099,632

We observe that approximately 500,000 new trades have been made in both smart contracts between 05/04/2020 and 07/08/2025 which is significantly less than the number of trades between 2017 (when the smart contracts were created) and 2020. It is a significant decrease compare to the numbers of trades between 2017 and 2020 which was expected regarding the situation explained previously (shutdown of EtherDelta and migration of IDEX to an other blockchain). Thus, we can assume that there will be no need to further expand this database due to the low benefits that can be gained from commercial washing and that its detection will therefore yield few results.

To run our algorithms, we also need the daily exchange rates of Ether to Dollar from Etherscan. The updated data is available on tubCloud [1], and the code on GitHub [2].

3.4 Results with Extended Dataset

As part of the project, we also extended the original dataset to include transactions from the decentralized exchange (DEX) IDEX covering the period 2020 to 2025, thereby bringing the data up to date. We reran the full detection pipeline, including the sliding window extension, on this expanded dataset.

Surprisingly, the results indicate only a modest increase in detected wash trading activity. Specifically, the number of detected wash trades rose from 186,413 to 195,320 - an increase of just 8,907

[1]https://tubcloud.tu-berlin.de/s/6WGL9HnK4QwJKRK
[2]https://github.com/alexschwind/wash-trading.git

Table 4: Comparison of previous and extended Dataset

	IDEX (to 2020)	IDEX (to 2025)
Number of relevant SCCs	199	210
Number of Wash Trades	186,413	195,320

trades over a five-year period. Similarly, the number of relevant strongly connected components (SCCs) rose from 199 to 210 — again, a relatively small change given the extended timeframe.

These findings suggest two important trends:

- Wash trading on this particular DEX has declined significantly in recent years, possibly due to increased scrutiny, improved detection tools, or declining interest in the platform.
- The overall trading activity on the exchange also appears to have shrunk, which may explain the low number of new suspicious clusters detected despite five additional years of data.

Taken together, the results imply that wash trading is no longer a widespread or urgent problem on this exchange, at least in terms of observable behavior in public transaction data. However, this also highlights the value of automated detection tools: had the problem persisted or evolved, the system would have been capable of detecting it efficiently and at scale.

4 Conclusion

In this project, we successfully reproduced and significantly improved the performance of a wash trade detection algorithm targeting decentralized exchange activity. By re-implementing the algorithm in Python with efficient libraries and applying parallelization to both the graph construction and volume analysis stages, we achieved a 5.5× reduction in total runtime, bringing execution time down from 45 minutes to 9 minutes. These optimizations made it feasible to explore algorithmic enhancements that were previously too costly to compute.

One of the most impactful improvements was the introduction of a sliding window mechanism, which enabled the detection of wash trades that span overlapping time intervals. This change resulted in a 13% increase in detected wash trades, demonstrating that the new approach not only improves performance but also enhances the accuracy and completeness of detection.

We also extended the dataset to include five additional years (2020–2025) of transaction history from IDEX. As expected, this broader dataset yielded only a small increase in suspicious activity, with detected wash trades rising from 186,413 to 195,320 and strongly connected components increasing from 199 to 210. This suggests that wash trading on IDEX has declined significantly in recent years, both in frequency and in scale, which could be explained by the migration to the new blockchain and thus, the low activity on the Ethereum blockchain. It is important to note that this decline may not necessarily indicate a systemic reduction in wash trading, but rather a shift in venue to newer protocols and platforms.

During experimentation, we also uncovered inconsistencies in the detection algorithm, particularly a sensitivity to the order in which transactions are processed. This observation highlights a need for algorithmic robustness, as detection results can vary depending on the sorting or grouping strategy applied. Addressing this instability—potentially by enforcing canonical processing order or modifying the balance calculation logic—remains an important direction for future work.

Due to time and format constraints, the analysis was limited to IDEX. However, the methods developed are general in structure and could be adapted to other decentralized exchanges similar EtherDelta with appropriate preprocessing. Future extensions may include cross-platform comparisons, real-time detection pipelines, or deeper behavioral modeling of suspicious trading clusters over time. However, given that EtherDelta is no longer operational and exhibits negligible new activity aswell as orderbook-services with a specfic seller and buyer are mostly deprecated, the application of these methods to similar exchanges is primarily of historical interest and unlikely to yield insights into contemporary wash trading dynamics.

In summary, this project demonstrates that with targeted algorithmic and system-level optimizations, it is possible to scale complex blockchain analytics tasks while simultaneously improving their analytical depth, runtime efficiency, and relevance.

5 Future work

This project opens several avenues for further investigation. The decentralized trading landscape has shifted towards Automated Market Makers like Uniswap and Layer-2 solutions, while orderbook-based DEXs such as EtherDelta have become inactive and IDEX has evolved to support new trading mechanisms and chains. As such, future research must prioritize adapting detection algorithms to these new architectures, where transaction patterns and the nature of counterpart interactions differ fundamentally from traditional models. Detecting wash trading in AMM-based exchanges remains an open question – since trades involve liquidity pools rather than direct account-to-account transfers, the patterns of wash trading may differ. Future research could adapt the graph-based approach to AMMs by, for example, identifying round-trip token swaps that return funds to the originator through a series of pool transactions. The original authors noted the promise of investigating newer DEX models, and our extended methodology could serve as a starting point for catching wash trades in those contexts. Similarly, cross-chain and Layer-2 DEXs, like Polygon, could be examined by porting the algorithm to their transaction logs – potentially uncovering if wash trading migrated to other ecosystems as Ethereum DEX activity declined.

Another promising direction is to evolve this batch detection process into a real-time monitoring system. Since our optimized pipeline can process millions of trades in minutes, one could envision running it on streaming data or with daily updates. In a live setting, the algorithm could flag suspicious trading rings soon after they occur, enabling quicker response by exchanges or regulators. To achieve this, we would need to handle continuous data ingestion and perhaps maintain rolling graph structures in memory. Our work on incremental volume calculation is a step toward this; a fully streaming implementation might use a sliding window that moves forward in time as new blocks arrive, continuously updating SCCs and checking for emerging wash cycles. Realizing real-time wash trade detection would greatly enhance practical impact – turning a forensic tool into a preventative surveillance mechanism.

References

[1] Friedhelm Victor and Andrea Marie Weintraud. [n. d.]. Detecting and Quantifying Wash Trading on Decentralized Cryptocurrency Exchanges. 23–32. https://doi.org/10.1145/3442381.3449824 arXiv:2102.07001 [cs]