The Power of Default: Measuring the Effect of Slippage Tolerance in Decentralized Exchanges

Nir Chemaya[1], Dingyue Liu[1],
Robert McLaughlin[2], Nicola Ruaro[2], Christopher Kruegel[2], and Giovanni Vigna[2]

[1] Department of Economics, University of California, Santa Barbara
[2] Department of Computer Science, University of California, Santa Barbara

Abstract. In recent years, we have seen the growth of decentralized finance (DeFi), an ecosystem of financial applications and protocols that enable complex, automated, permissionless financial transactions in blockchains (such as Ethereum). We examine decentralized exchanges (DEX), a key DeFi component that facilitates token swaps. DEX prices update continuously and automatically after each swap, creating price shifts for users as their swaps (trades) wait to execute. Users protect themselves from these price shifts by setting a *slippage* tolerance, which represents the maximum acceptable price increase. This setting is a double-edged sword: lenient tolerance can be exploited through *sandwich attacks*, which cost the ecosystem over $100 million annually, but stricter tolerance may cause unnecessary failures. We perform a large-scale measurement of the impact of slippage tolerance settings on the health of the Uniswap and Sushiswap DEX ecosystems. To this end, we examine a recent change in Uniswap's default slippage setting, which aimed to mitigate sandwich attacks without increasing the likelihood of transaction failures. This change removed the prior, static default – 0.5% – in favor of one dynamically computed for each transaction based on market conditions so that sandwich attacks are less profitable. We find that, overall, Uniswap's new default slippage setting leads to a substantial reduction in Uniswap traders' losses, approximately 54.7%. The effect is even more pronounced 90% when we only consider traders who followed the default settings. Additionally, we propose some directions for further improving of the default settings.

Keywords: Blockchain · DeFi · Automated Market Makers · Default Effect.

1 Introduction

Nudge theory suggests that small environmental features can capture attention and influence behavior and decision-making, as argued by Thaler et al.[21]. The phenomenon of the *default effect*, whereby people tend to stick with the default decision, is a well-known example of this theory. This effect occurs when individuals are presented with a choice, and the decision architecture suggests a default option. When individuals do not actively choose an alternative, the default option is the one they ultimately end up with. As a result, the likelihood of that particular decision being selected is higher in comparison to the other available options. Given that the default decision can significantly impact people's choices, policymakers must select

the optimal default to promote greater social welfare. There is a wealth of evidence on the default effect, particularly in the realm of financial decisions. To enhance welfare, policymakers have implemented measures such as adjusting default choices, as demonstrated by the "Save More Tomorrow" program [20].

We can find many default settings in many different Decentralized Finance (DeFi) platforms. For example, many Decentralized Exchanges (DEX) platforms have a default slippage tolerance setting for traders. This slippage tolerance protects the user in case there is a change in the expected exchange rate, i.e., the smart contract will execute the transaction only if the traders obtain, at a minimum, a specified quantity of tokens. Additionally, it can protect users from "sandwich attacks," which is a price manipulation tactic wherein an attacker strategically influences market prices to exploit traders on DEX platforms. We will describe this in detail in the following sections.

The slippage tolerance setting must be chosen carefully. By choosing higher slippage tolerance values, traders increase the potential profit for sandwich attacks, which makes them more vulnerable to such attacks. Consequently, it is imperative that traders choose a slippage tolerance that is not too high. Conversely, choosing lower slippage tolerance values raises the likelihood that the transaction fails because of price movement caused by normal, but unanticipated, trades from other users. When the the slippage tolerance threshold is exceeded, no swap will be performed, but the user must still pay gas fees. Therefore, traders must carefully consider these trade-offs when deciding on their preferred slippage levels.

Ethereum currently carries the highest Total Value Locked (TVL) among blockchains in the DeFi ecosystem, and on which Uniswap is the most prominent DEX platform, as measured by TVL [9]. Initially, Uniswap set the default slippage tolerance at 0.5%. However, on March 16th, 2023, the Uniswap protocol implemented a modification to its default slippage tolerance mechanism. This new approach takes into account estimated network fees and the size of traders' transactions to compute a customized slippage for each trade [26]. Depending on the specific trade and market conditions, this new default can range from 0.1% to 5%. The primary objective of this modification is to reduce the impact of sandwich attacks that leverage price manipulation. Fortunately, other DEX protocols like Sushiswap [19], have maintained the default slippage tolerance at 0.5%. This allows us to use Sushiswap as a control when we are conducting our analysis for the impact of Uniswap's new default settings.

To understand how this dynamically determined slippage tolerance impacts the ecosystem, we performed a large-scale measurement. More precisely, we analyzed all the transactions executed on Uniswap and Sushiswap protocols on the Ethereum blockchain in the month of March 2023 – amounting to approximately 5 million swap transactions – from which we infer the slippage tolerance setting of 1,187,141 Uniswap and 19,111 Sushiswap swaps.

The results of our analysis using new measurement approaches provided several insights, which are the contributions of this paper:

1. We developed a methodology for utilizing data from the Ethereum public transaction queue (the *mempool*) to infer traders' decision-making regarding slippage in trading. By employing this technique, we examine the existence of a default effect in DEX platforms. Our findings reveal that 24% of Uniswap transactions and

67% of Sushiswap transactions adhere to the default slippage tolerance offered by the protocols. This demonstrates the substantial impact of the protocols' default settings on trading behavior.
2. We evaluate the effectiveness of Uniswap's recently introduced slippage mechanism in terms of the benefits it provides to the traders' welfare. Notably, our findings indicate that the adoption of Uniswap's new default slippage setting leads to a substantial reduction in traders' losses by approximately 54.7%. We conducted a robustness check to ensure that this effect is primarily observed in users who follow the default, i.e., defaulters. We found that this is indeed the case, with an approximate 90% reduction in losses for defaulters' welfare. Considering that default users are still suffering losses from sandwich attacks, this underscores the potential for further improvement of the default settings.

2 Background

2.1 Decentralized Exchanges

DEX platforms have emerged as a new option for traders in the cryptocurrency market, enabling them to engage in decentralized trading activities without relying on the conventional centralized order book mechanism such as the one that Coinbase operates [7]. Many DEX platforms use liquidity pools, which are smart contracts that allow agents to provide liquidity in the form of tokens or assets on the blockchain. Most pools have liquidity for two tokens/assets and enable users to exchange between them. Traders exchange tokens or assets from these pools using a pricing rule specified in the smart contract code. One of the most commonly used pricing rule in these liquidity pools is the Constant Product Market Maker (CPMM), which is determined by the supply of tokens or assets in the pool [28]. Liquidity providers are incentivized to engage in these pools by receiving a fee from each swap action, which typically ranges from 0.01% to 1%, depending on the protocol and the tokens/assets in the pool. Considering the predominant trading volume observed on the Ethereum blockchain, this work focuses exclusively on the analysis of DEX data on the Ethereum blockchain. Uniswap is currently the largest DEX protocol in DeFi, with a daily volume of roughly $1 billion and total liquidity of $2 billion [24] on the Ethereum blockchain.[3]

2.2 Sandwich Attacks

Despite their popularity, users of DEX platforms regularly suffer from *sandwich attacks*, an issue which is endemic to the ecosystem. A sandwich attack is a price manipulation tactic wherein an attacker strategically influences market prices, which forces unassuming traders into accepting higher-than-necessary prices.

In Ethereum, pending transactions are first sent into a public queue, the *mempool*, before they are executed. A malicious user (Mallory) can observe these pending transactions and exploit them using the sandwich attack pattern. During a sandwich attack, Mallory monitors public transactions queued in the mempool and strategically

[3] Data source: https://uniswap.org/ on March 16, 2023 (only Ethereum).

inserts two of her own transactions immediately before and after the victim's transaction. These transactions are commonly referred to as front-running and back-running, respectively. Attackers, by using this tactic, can gain a massive profit depending on the trade size and market conditions. The total annual profit extracted by sandwich attackers is approximately over $100 million annually [18].

For instance, consider a scenario where Alice intends to trade 10 of token A for 1 token B, which, in this example, is the expected exchange price in the liquidity pool if her transaction is executed next. Alice's transaction is sent to the mempool and awaits execution by a block producer. Mallory observes Alice's pending transaction, and strategically inserts a large swap immediately before Alice's, which raises the exchange's price significantly. Alice's transaction then executes after Mallory's, but with a worse price due to the CPMM pricing rule (for example, assume that she now receives only 0.8 B tokens). Finally, Mallory will execute another large swap in the opposite direction of exchange, restoring the un-manipulated exchange price. This results in Mallory earning a profit of 0.2 B tokens at Alice's expense. A prototypical sandwich attack is shown in more detail in Fig. 6 in Appendix A.4. Mallory may also choose to generate "multi-meat" sandwich attacks, wherein a single attack, i.e., a pair of front-running and back-running transactions, targets multiple victims simultaneously. This is illustrated in Fig. 7 in Appendix A.4.

2.3 Related Work

Extensive research has been conducted on DEX platforms from various perspectives, providing valuable insights into their functioning. Notably, several studies (Lehar and Parlour [13]; Barbon and Ranaldo [3]) have focused on comparing centralized exchange (CEX) and decentralized exchange (DEX) platforms, analyzing crucial factors such as liquidity provision, absence of arbitrage, price efficiency, and transaction costs. Additionally, a significant body of literature (Park [17]; Capponi and Jia [4]) has been devoted to investigating the Constant Product Market Maker (CPMM) mechanism, examining its properties, and identifying conceptual flaws.

Among the identified flaws, one prominent concern highlighted in Park's work [17] is the vulnerability of traders to sandwich attacks. Numerous papers have delved into the concept of sandwich attacks, discussing their empirical existence (Daian et al. [8], Zhou et al. [30], Lehar and Parlour [14], Züst [31], Qin et al. [18] and Torres et al. [23].)

The work by Heimbach and Wattenhofer [12] is particularly relevant to our research. They introduced a model called the "Sandwich Game," which is based on game theory and allows traders to protect themselves against sandwich attacks by selecting an optimal slippage protector. The authors propose a dynamic slippage approach, wherein the algorithm calculates an optimal slippage protector that reduces the probability of sandwich attacks without significantly increasing the likelihood of transaction failures. Building upon this, our paper focuses on the actual slippage decisions made by traders and their default effects within this environment. Specifically, we explore the behavior of users adhering to the default slippage decision and examine the effects of altering this default within the Uniswap protocol. In line with the spirit of Heimbach and Wattenhofer's model, Uniswap transitioned from a constant

slippage approach to a more dynamic one. We shed light on the effectiveness of this new slippage methodology and propose potential avenues for further improvement.

3 Model

In this section, we summarize some key properties of the Constant Product Market Maker (CPMM) pricing formula, which is used by several DEX applications, including Uniswap and SushiSwap. This will allow us to formalize traders' slippage decisions in DEX CPMM applications and to provide a straightforward model for estimating slippage decisions based on mempool data.

3.1 Transaction Model

Let's consider a liquidity pool that contains x tokens of token $\mathbf{X}$ and y tokens of token $\mathbf{Y}$ (following the notation of [3].) The CPMM pricing rule specifies that for any time t, the product of the available tokens ($\mathbf{X}$ and $\mathbf{Y}$) in the pool equals a constant k, which can be expressed as

$$x_t y_t = k \tag{1}$$

Let f denote the percentage fee that is subtracted from each swap's payment and remitted to the liquidity providers. Then, $\varphi = 1 - f$ is the percentage left for the trader to swap. If at time $t+1$ a trader wants to swap Δx quantity of tokens $\mathbf{X}$ for $\mathbf{Y}$ tokens, we can calculate how many tokens $\mathbf{Y}$ that trader will receive, $\Delta(y)$. CPMM states that

$$k = (x_t + \varphi \Delta x)(y_t - \Delta(y)) \tag{2}$$

For illustration, the amount $\Delta(y)$ received when paying $\Delta(x)$ is:

$$\Delta(y) = y_t \frac{\varphi \Delta x}{x_t + \varphi \Delta x} \tag{3}$$

DEX applications commonly allow users to specify either Δx, the exact amount to pay, or Δy, the exact amount desired. The unspecified quantity is computed on-demand, according to the equations above. In order to disambiguate the user-specified value and the computed value, we write the given value as either Δx or Δy and the computed value as either $\Delta(x)$ or $\Delta(y)$.

To protect the trader from sandwich attacks and price manipulations, DEX platforms, such as Uniswap, allow traders to define a maximum slippage percentage ϕ. From this percentage, the application derives either a minimum amount received, $\underline{\Delta(y)}$, or a maximum amount paid, $\underline{\Delta(x)}$. This means that if there is any change in the expected exchange rate due to a change in the pools – possibly due to sandwich attacks – the smart contract will execute the transaction only if the trade meets this threshold. For illustration, the minimum amount received ($\underline{\Delta(y)}$) is computed as:

$$\begin{aligned} \underline{\Delta(y)} &= (1+\phi)^{-1} \Delta(y) \\ &= (1+\phi)^{-1} * y_t * \frac{\varphi * \Delta x}{x_t + \varphi * \Delta x}. \end{aligned} \tag{4}$$

This CPMM pricing formula is used by both Uniswap V2 and Sushiswap [1]. Some liquidity pools in Uniswap use the newer V3 model [2], which is slightly more nuanced. This model allows liquidity providers to bound the price range in which their liquidity is usable by the pool – if the price moves outside of the user's configured range, their deposits are removed from market-making activities.

A user who wants to perform a swap on a DEX typically proceeds as follows: The user first accesses the DEX user interface – in the case of Uniswap and Sushiswap, this UI is a (regular) web application. The user specifies the token they wish to sell, the token they wish to receive, and either an exact amount paid, Δx, or an exact amount to receive, Δy.

At this point, the user can either accept the default option and follow the slippage decision suggested by the DEX protocol, or manually specify their preferred slippage percentage, ϕ. The DEX application will then compute $\underline{\Delta(y)}$ or $\underline{\Delta(x)}$ using the current DEX state.

Finally, the user reviews, signs, and transmits the transaction to the Ethereum mempool, where it queues for execution. This is typically done with the help of a browser extension, such as Metamask [16]. Note that the transaction includes the computed value $\underline{\Delta(y)}$ or $\underline{\Delta(x)}$, but the value ϕ is not encoded within the transaction. In the following section, we explain how we determine (estimate) the value ϕ by observing the Ethereum mempool.

3.2 Estimating the Slippage Tolerance Decision

A user's choice of ϕ has an important impact on the transaction's execution. However, this value is not broadcast to the network. Thus, we use the following method to recover the user's original choice.

We derive the slippage percentage ϕ using Equation 4:

$$\phi = \Delta(y)/\underline{\Delta(y)} - 1 \tag{5}$$

While the values of $\underline{\Delta(y)}$ and $\underline{\Delta(x)}$ can be immediately read from the transaction, the values of $\Delta(y)$ and $\Delta(x)$ are not encoded within the transaction.

For the CPMM pricing rule, computing $\Delta(y)$ and $\Delta(x)$ requires knowing what the state of the DEX was *when the application generated the swap transaction.*

Consequently, we first estimate *when* the transaction was generated. By monitoring the Ethereum mempool, we record the timestamp t at which the transaction was first broadcast.[4] We then look up the DEX state as of timestamp t, which we use to compute either $\Delta(x)$ or $\Delta(y)$, and finally compute ϕ.

[4] Prior work observed that the amount of time to propagate a transaction across the peer-to-peer mempool network is small (200 milliseconds) compared to the block interval (12 seconds) [27]. As we only require accuracy to within the block interval, time t will suffice as an estimate for the transaction generation time.

4 Empirical Evaluation

4.1 Experimental Setup

To perform our study, we require two types of data: first, the timestamp when a transaction entered the public mempool, and, second, a complete record of all Uniswap and Sushiswap DEX swaps that were executed on the blockchain.

To address both needs, we run an Ethereum node in archive mode using Go-Ethereum [11] version 1.11.2. The node is hosted in the western United States and is served by a high-bandwidth Internet connection. In order to increase mempool visibility and decrease transaction propagation latency, we increased the number of peer-to-peer connections from 100 to 500.

We ran our data collection from March 1, 2023, 00:00 UTC to April 1, 2023, 00:00 UTC, and examined all 33,108,538 transactions within blocks 16,730,072 through 16,950,602. We used a simple script to record the arrival time of each transaction in the mempool, which satisfies our timestamping needs outlined in Section 3.2.

Swaps generated through the Uniswap and Sushiswap web apps begin by invoking a *router* contract. As implied by the name, the router contract routes a user's swaps through the protocol, making use of at least one, but possibly several, liquidity pools. Importantly, the router enforces a DEX user's slippage tolerance setting. Third-party DeFi applications and bots commonly perform swaps by directly interacting with liquidity pools, foregoing the Uniswap or Sushiswap router entirely.

We find all router-based swaps by retrieving all confirmed transactions within the study window, which we then filter down to only transactions sent to one of the known router contracts. We then use our knowledge of the router's application binary interface (ABI) to extract the swap's parameters – Δx or Δy and $\underline{\Delta(y)}$ or $\underline{\Delta(x)}$. We find that 81.7% of swaps on Uniswap use exactly one liquidity pool, so, for simplicity of calculation, we only consider router-based swaps that use exactly one liquidity pool. We also infer the total dollar value of the amount swapped by aggregating several on-chain price oracles.

4.2 Mempool Observation

Our Ethereum node observed 32,586,069 transactions queued in the mempool. After filtering these down to only transactions within the study window that invoke the Uniswap or Sushiswap routers, we uncover 1,640,182 router-based transactions. We present the arrival rate in Figure 1.

Our Ethereum node captured 1,617,820 (98.6%) of the router-based transactions while they were queued for execution. The remaining router-based transactions were not observed in the mempool, but did get included in the blockchain. This may be due to either failure to propagate through the mempool network, or use of a private transaction relayer such as Flashbots-Protect [10]. Transactions broadcast through private relayers circumvent the mempool and are privately sent directly to block producers. This shows that our mempool observation is broadly effective and, thus, our analysis is truly representative of the ecosystem.

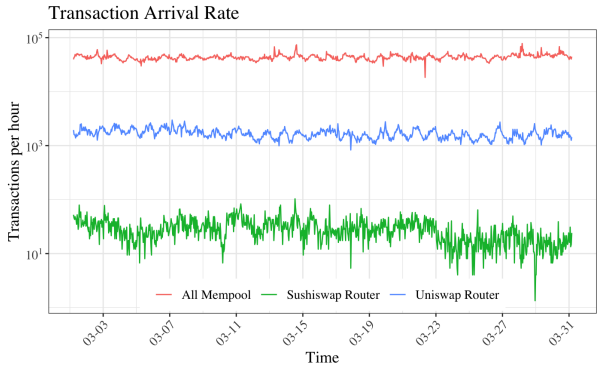

Fig. 1: Observed hourly transaction rate. Note the use of a log scale to show detail.

4.3 Inferring Slippage Tolerance Decisions

Router-based swap transactions include either Δx (the exact amount the user would like to pay) or Δy (the exact amount the user would like to receive).[5]

We begin with the subset of transactions observed in the mempool – 1,586,378 Uniswap and 31,442 Sushiswap. The following transactions are then also removed. First, for simplicity, we discard transaction that use multiple liquidity pools – 294,412 Uniswap and 8,103 Sushiswap. Second, we discard transactions that transfer tokens which interfere with the liquidity pool the trader would like to use. For illustration, some tokens charge a tax upon each transfer from one address to another. A token can then immediately sell the tax on Uniswap to immediately convert it a more stable currency, which is deposited into a treasury. When this occurs it is difficult to infer the slippage, as we would need to reason about each token's arbitrary behavior. We discard any transactions which demonstrate this behavior – 64,682 Uniswap and 0 Sushiswap. Third, we discard transactions that are not using the router to perform a swap – for example, providing liquidity to a pool, buying NFTs, etc. – 173 Uniswap and 3,611 Sushiswap. Fourth, we discard malformed transactions – 279 Uniswap and 129 Sushiswap. This leaves 1,231,615 Uniswap and 19,765 Sushiswap transactions.

In total, we can infer the slippage decisions for 1,187,141 (96.4%) Uniswap and 19,111 (96.7%) Sushiswap router-based transactions observed in the mempool. In swap transactions for which slippage inference failed, we were unable to derive an expected swap price, $\Delta(y)$ or $\Delta(x)$, at the time of transaction generation. This was either because the pool did not have enough liquidity to support the user's desired quantity of tokens, or because, upon inspection, our timestamp of the transaction's arrival in the mempool fell after the timestamp of the block in which it was executed – possibly due to network latency or clock drift. We draw the most popular slippage settings over the time window in Figure 2, trimmed to 0% to 2% to show detail. For a wider range of percentages, see Figure 5 in the Appendix A.3.

[5] We find that it is much more popular to specify Δx – accounting for 91.0% and 96.3% of all Uniswap and Sushiswap router-based transactions, respectively.

We immediately observe the following two oddities. First, we occasionally infer a *negative* slippage setting – this accounts for 2.6% of Uniswap and 4.4% of Sushiswap transactions. To illustrate how this might occur, consider a transaction that specifies Δx. In this situation, we use Equation 5 to derive the slippage. Negative values occur whenever $\underline{\Delta(y)} > \Delta(y)$ – i.e., whenever the user requires a minimum amount received that exceeds what the DEX can remit. These transactions have a very low success rate – only 6.9% and 14.7% for Uniswap and Sushiswap, respectively. Second, we occasionally infer slippage settings that are extremely high (50% and over) – however, the Uniswap and Sushiswap user interfaces both cap slippage to 50%. This accounts for 1.0% of Uniswap and 0.6% of Sushiswap transactions. Both of these oddities may plausibly indicate either a flaw in the transaction generation software, an incorrectly inferred transaction generation time, or simply spam. Nevertheless, these transactions have relatively small prevalence, and are likely not representative of a web app user – so we are confident that their exclusion does not significantly impact the results.

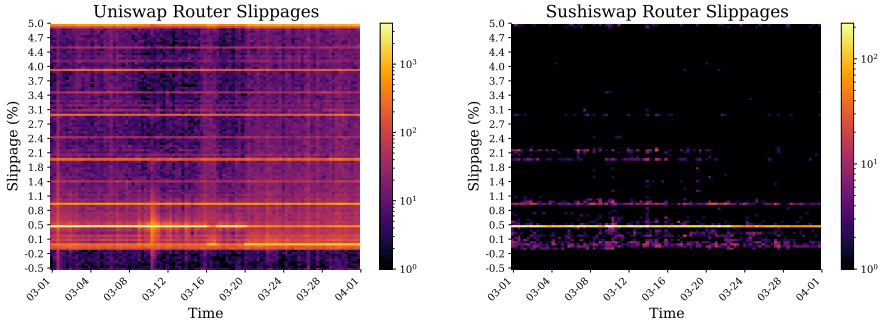

Fig. 2: Inferred router slippages over time. Color is drawn on a log scale to show detail. Each colored cell counts the number of swaps in a slippage range at a segment of time.

4.4 Default Effect on Slippage Tolerance

We now examine the *default effect* on slippage choices in Uniswap and Sushiswap exchanges. This can be seen intuitively in Figure 2: we can see a clear, bright line at the default slippage setting, 0.5%. This default was active for Uniswap up to March 16^{th}, and for Sushiswap throughout the entire study window.

Table 1 details the percentage of inferred slippage choices at various percentages. We see a preference for the default setting in both cases.

In the case of Uniswap, we also observe a preference for round numbers (10%, 5%, etc.), both in Table 1 and in Figure 2. Figure 5, in the Appendix A.3, plots a version of Figure 2 with a wider y-axis range, from 0% up to 25% – here we can also observe a clear round number preference.

We do not attempt to distinguish between human- and bot-originated transactions, so we cannot be sure of exactly which transactions are representative of human decision-making when faced with a default. However, we can observe a significant aggregate change in slippage settings after Uniswap altered its default on March 16, 2023.

Initially, the implementation was carried out in a soft manner, resembling a "testing period" that lasted from March 16th to the 20th. This testing period is evident in our data and can be observed in Figure 2. On March 16, there was a noticeable decrease in decisions with a 0.5% slippage rate. This is shown by the yellow line disappearing. Afterward, it appeared again for a few days during the soft launch, but then it disappeared once more.

	Slippage Setting					
	10%	5%	2%	1%	**0.5%**	0.1%
Uniswap [6]	7.3%	7.1%	2.2%	4.7%	**24%**	2.1%
Sushiswap	1.1%	1.0%	1.9%	5.1%	**67%**	2.5%

Table 1: Prevalence of various slippage settings in router-based swaps.

From March 21^{st} onward, after the default fully was changed, we see only 4.7% of Uniswap transactions use the old default: 0.5%. Moreover, we see a six-fold increase (to 11%) in the percentage of transactions using 0.1% slippage, a soft minimum in the new default's "auto" calculation. This gives confidence that users are, in fact, following the default.

Following this, we label which transactions are most likely using the default slippage setting. This must be done in two parts, before and after the default was changed. In the case of the former, the computation is straightforward: we label all transactions with a slippage between 0.45% and 0.55% as using the default. The latter case is more involved. In brief, Uniswap's default slippage setting is computed by taking the gas cost the user pays to perform the swap, in US dollars, divided by the transaction volume, in US dollars. The resulting value is then bounded to within the range [0.1%, 5%]. Both the numerator and denominator are subject to market fluctuations: gas fees are natively paid in Ether, not dollars, and the transaction volume is specified by the user in units of tokens. Uniswap converts both values to dollars by consulting the Uniswap DEX for spot prices. For each transaction, we derive an estimated default slippage setting by recreating the computation using historical data and market conditions. Then, we label a transaction as using the default if its inferred slippage setting is within 20% of our estimated default. The technical details of this approach are described in [25].

4.5 Traders' Welfare Post Modification of Uniswap's Default Slippage Setting

In the following analysis, we assess the adjustment made to the default slippage setting within the Uniswap decentralized exchange (DEX) protocol on March 16, 2023. Our main goal is to evaluate whether traders are benefiting or suffering from this modification, i.e., if their welfare is increased or decreased. This evaluation encompasses two key aspects: the losses of traders affected by sandwich attacks and the losses incurred due to unnecessary failed transactions, both of which are considerations when determining the optimal slippage tolerance.

[6] Before March 16, 2023.

To conduct this evaluation, we gathered data from both the Uniswap and Sushiswap protocols for the month of March 2023 – not limiting ourselves to only the "official" routers – which yielded approximately 5 million transactions. We divided our data analysis into three distinct time periods to assess the welfare impact of this new method.

The first period covers the time before the modification, from March 1st to 15th. The second period corresponds to the "testing period" from March 16th to 20th. Finally, the third period focuses on the time after Uniswap adopted the new slippage protector, from March 21st to 31st.

Notably, during our data collection period, Sushiswap maintained a fixed 0.5% default slippage option, whereas Uniswap implemented the dynamic one. This distinction enables us to use Sushiswap as a reference point for control purposes in our analysis, especially due to the major similarity of both platforms.[7]

We focused exclusively on pools with tokens that had high liquidity to avoid price inaccuracies that could affect our aggregate analysis. We discovered that tokens with low liquidity tend to have very inaccurate price oracles,[8] leading to erratic results when converting to US Dollar value ($). We further filter the data set to include only the most active pools that had swap activity every day in March. This approach enables us to minimize the influence of noise generated by pools that were only available before or after the change, and it minimizes the influence of bots conducting pump-and-dump schemes [5]. Finally, our filtered data covers approximately 2.25 million transactions, accounting for around 47% of the total transactions. It includes 824 unique pools, which is approximately 4.4% of the total unique pools 18,877. This indicates that the majority of the transactions on the protocols are conducted pools with high liquidity and consistent trading activity. A more detailed information about our swap data set can be find in Table 9 in Appendix A.2

4.6 Loss from Sandwich Attacks

Our detection method for sandwich attacks – detailed in Appendix A.1 – draws on earlier studies by Lehar and Parlour [14], Züst [31], Qin et al.[18], and Torres et al. [23].

Using this method we identify total of 36,158 successful attacks in our data, with 33,612 attacks occurring on Uniswap and 2,546 attacks on Sushiswap. Table 2 presents the quantity and daily rate of successful attacks detected.

Next, we analyze the financial impact and quantify the losses incurred due to the attacks on both Uniswap and Sushiswap. By examining the value of assets lost during these attacks, we can gain a comprehensive understanding of the implications of the default slippage change and its effectiveness in mitigating losses.

We then conducted the following analysis in order to assess the financial impact of the attacks. For each attacked swap, we calculated the difference between the trader's outcome with and without the attack, and standardized to a dollar amount. For our

[7] Sushiswap is a forked version of Uniswap, and both platforms work very similarly. More information can be found here:[15,29]

[8] We calculate the minimum Total Value Locked (TVL) within the specified window for each token. We then select only pools that have both tokens ranked in the top 1,000 from this list, which we classify as tokens with high liquidity.

analysis, we employed the V2 model due to its straightforwardness. When examining a Uniswap V3 pool, we approximated its pricing function by keeping its instantaneous Constant Product Market Maker (CPMM) bonding curve fixed, meaning we did not make any liquidity adjustments to account for price impact. Prior analysis shows that this approach yields highly accurate results [6]. Collectively, all attacks on Uniswap and Sushiswap resulted in a total loss of $16,584,438 for traders in March. On average, each swap that was attacked incurred a loss of $458.7, with a median loss of $69.6. The breakdown of these loss figures by DEX and time period can be found in Table 3.

	Time Period	Number of Attacks Detected	Average Attacks Per Day
Uniswap	Mar 1st - 15th	23,299	1,553.27 (100%)
	Mar 16th - 19th	3,214	803.5
	Mar 20th - 31st	7,099	591.58 (38.09%)
Sushiswap	Mar 1st - 15th	1,297	86.47 (100%)
	Mar 16th - 19th	464	116.0
	Mar 20th - 31st	785	65.42 (75.66%)

Table 2: Attacks Detected.

In order to better evaluate the effect of the implementation of dynamic slippage, we aimed to directly compare Uniswap and Sushiswap across all periods, while mitigating the influence of differing trade sizes in different periods. To achieve this, we standardized the loss by the total value swapped. For each day in March, we calculated the total loss from attacks per $100 swapped on each DEX, which we plot in Figure 3.

	Time Period	Mean	Percentile			Daily Total
			25^{th}	50^{th}	75^{th}	
Uniswap	Mar 1st - 15th	$565	$33	$89 (100%)	$298	$877,936 (100%)
	Mar 16th - 19th	$292	$21	$47	$138	$234,874
	Mar 20th - 31st	$277	$23	$51 (56.8%)	$134	$163,807 (18.7%)
Sushiswap	Mar 1st - 15th	$250	$22	$50 (100%)	$146	$ 21,596 (100%)
	Mar 16th - 19th	$119	$16	$30	$73	$ 13,764
	Mar 20th - 31st	$167	$18	$36 (72.9%)	$75	$ 10,934 (50.6%)

Table 3: Loss from Attacks.

The y-axis in the figure represents the loss per $100 swapped, where a value of 0.02 signifies that the loss from an attack would account for 0.02% of the swapped value. On the left-hand side of the figure, we have Uniswap, while Sushiswap is on the right-hand side. It is important to note that the range of the y-axis differs between the two plots, with values for Uniswap mostly below 0.05% and Sushiswap reaching up to 0.25%.

The colored lines in the figure represent the weighted mean for each period. The red line corresponds to the pre-launch period from March 1^{st} to March 15^{th}, while the purple line represents the period after the full launch. For Sushiswap, the two colored lines are nearly level, suggesting no significant impact from the change of the default

slippage setting on Uniswap. However, a noticeable gap is observed for Uniswap. This gap is statistically significant as shown with the non-overlapping dashed line, which represent the 95% confidence interval. This result confirms that after the deployment of dynamic slippage, traders experienced reduced losses due to sandwich attacks, resulting in a more favorable trading environment on Uniswap.

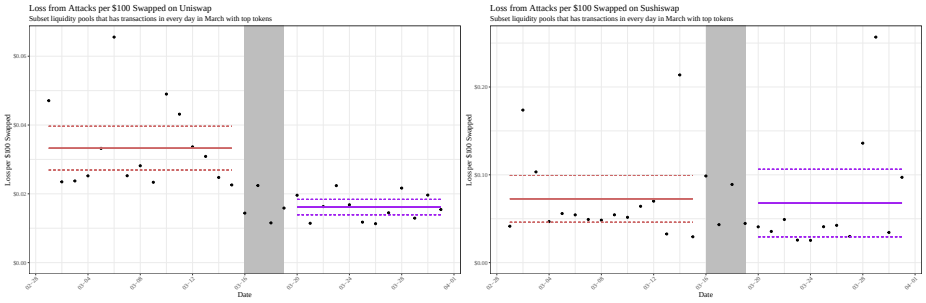

Fig. 3: Loss from sandwich attacks per $100 on Uniswap vs Sushiswap.

4.7 Loss from Unnecessary Failures

We further investigated the impact of dynamic slippage from the perspective of additional losses resulting from unnecessary failures due to potential tighter slippage. By scanning the mempool information, we identified all transactions in our data set that failed due to triggered slippage protection, recording the gas losses incurred from these swaps. An overview of the number of swaps that failed due to triggered slippage and the total gas losses in March can be found in Table 4.

	Failed Swaps	Failed due to slippage protector	Gas Loss
Uniswap	18,136	7,928 (43.7%)	$77,813.8
Sushiswap	876	777 (88.7%)	$ 1,643.0
All	19,012	8,705 (45.8%)	$79,456.8

Table 4: Failed Swaps that triggered Slippage Tolerance.

Interestingly, losses due to attacks measures 200 - 300 times greater than gas paid upon failure.

4.8 Cumulative Impact on Traders' Welfare

Finally, we calculate the total welfare loss, i.e., taking into account both the loss from sandwich attacks and gas losses for unnecessary failed transactions, for each protocol before and after the modification of Uniswap as shown in Table 5. We find that overall, Uniswap's new default slippage setting leads to a substantial reduction in traders' losses (54.7%) while in Sushiswap, the reduction is only 4.9%.

Next, we investigate the impact on users who follow the default, i.e., defaulters.

We managed to classify defaulters for 20,477 Uniswap and 7,919 Sushiswap transactions. [9] We will use these groups of transactions that follow the default settings to check if we can observe the same effect or even a greater one in the Uniswap default users. For robustness checks, we will compare it to Sushiswap default users as our control group. This allows us to conduct the same analysis as before, but only on the default users, as shown in Table 5. Before the change, default users on both platforms experienced a similar loss, approximately 0.13% of transaction value, which is much higher than the losses of the average user, which are 0.037% in Uniswap and 0.081% in Sushiswap. This implies that attackers may target these 0.5% default users regardless of which protocol they are using. The results show that the effect on Uniswap defaulters was much greater, with a 90% reduction in traders' losses, while for Sushiswap defaulters, there was almost no change at all, with only a 4.9% reduction. These findings suggest that DeFi protocols, such as Sushiswap, should consider changing their default settings to a new mechanism similar to the one utilized by Uniswap.

Loss per $100	Users	Before	After	Change
Uniswap	All	0.0371	0.0168	54.7% decrease
	Defaulter	0.1366	0.0137	**90.0% decrease**
Sushiswap	All	0.0816	0.0706	13.5% decrease
	Defaulter	0.1369	0.1302	4.9% decrease

Table 5: Cumulative Impact on Traders' Welfare

4.9 Potential For Further Improvement

Selecting the correct slippage tolerance for a transaction requires balancing the risk of overpaying and the risk of failure due to unexpected, small price movement. We have shown that Uniswap's new automatic default slippage tolerance setting improves the risk of the former with little-to-no additional risk to the latter. Research on further improving this setting must answer the following questions:

1. Exactly how close to zero can the slippage setting be set before being *too small*? We provide a partial answer to this through Figure 4, which shows the CDF of the realized slippage percentage in which each transaction incurred. The figure shows that 81.0% of all transactions actually experienced *zero slippage* – i.e., no unexpected price movement occurred whatsoever. Moreover, this percentage grows slowly. This suggests that unexpected price movement may be less common than previously believed, potentially allowing for even smaller slippage settings than the soft minimum of Uniswap's new default (which is 0.1%).

[9] We were only able to classify transactions for which we managed to infer their slippage tolerance decisions (as described in section 4.3), and these were executed in pools that met our consistency and liquidity availability restrictions, (as described in section 4.5).

2. Which dynamic slippage mechanism is the optimal one to achieve the highest welfare for traders? The idea of having a dynamic slippage mechanism to determine a customized slippage for each trade is important and could indeed improve traders' wealth and reduce sandwich attacks, as we showed in our analysis. Given that default users are still experiencing losses from sandwich attacks with little-to-no additional cost from unnecessary transaction failures, it can highlight the potential for future improvement.

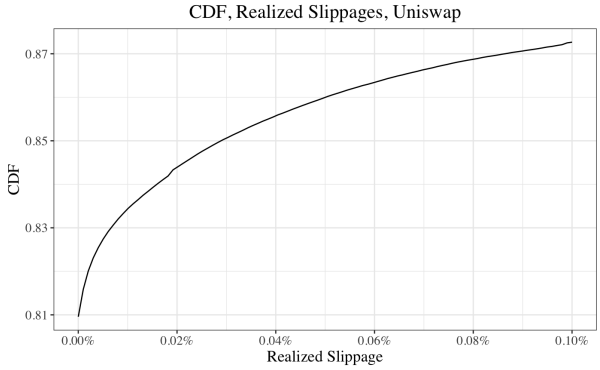

Fig. 4: Realized slippage in successful Uniswap transactions.

5 Conclusions

In this study, we examined the impact of slippage tolerance settings on the health of the decentralized exchange (DEX) ecosystem, focusing on the default slippage tolerance offered by various platforms.

We observed that a significant proportion of transactions, 24% in Uniswap and 67% in Sushiswap, respectively, adheres to the default slippage settings provided by the protocols. This indicates the substantial impact of default options on users' choices and highlights the importance of selecting optimal defaults to promote greater social welfare.

Additionally, we evaluated the effectiveness of Uniswap's recent modification to its default slippage mechanism. By analyzing traders' data, we assessed the benefits of the new default setting in terms of traders' welfare. Our results indicated a substantial reduction in traders' losses, approximately 54.7% (and 90.0% for default transactions). This demonstrates the potential of dynamic slippage mechanisms to improve the overall trading experience and protect users from malicious activities. Considering that default users are still encountering losses due to sandwich attacks, without incurring significant additional costs from unnecessary transaction failures, this suggests that there is room for improvement in the default settings.

The significance of our research highlights the necessity for rigorous academic exploration of default settings within the DEX and decentralized finance (DeFi) domains. By comprehending the implications of default options and introducing innovative approaches to enhance security and traders' welfare, we can actively contribute to the development and long-term sustainability of the DeFi ecosystem.

References

1. Adams, H., Zinsmeister, N., Robinson, D.: Uniswap v2 core (2020)
2. Adams, H., Zinsmeister, N., Salem, M., Keefer, R., Robinson, D.: Uniswap v3 core (2021)
3. Barbon, A., Ranaldo, A.: On the quality of cryptocurrency markets: Centralized versus decentralized exchanges. arXiv preprint arXiv:2112.07386 (2021)
4. Capponi, A., Jia, R.: The adoption of blockchain-based decentralized exchanges. arXiv preprint arXiv:2103.08842 (2021)
5. Cernera, F., Morgia, M.L., Mei, A., Sassi, F.: Token spammers, rug pulls, and sniper bots: An analysis of the ecosystem of tokens in ethereum and in the binance smart chain (BNB). In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 3349–3366. USENIX Association, Anaheim, CA (Aug 2023), https://www.usenix.org/conference/usenixsecurity23/presentation/cernera
6. Chemaya, N., Liu, D.: Analyze uniswap v3 data using v2 methods. Available at SSRN 4372372 (2023)
7. Coinbase: Coinbase. https://www.coinbase.com/legal/trading_rules (2023)
8. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 910–927. IEEE (2020)
9. defillama: Dexes tvl rankings. https://defillama.com/protocols/dexes/Ethereum (2023)
10. Flashbots: Flashbots protect. https://docs.flashbots.net/flashbots-protect/overview (2023)
11. Foundation, E.: go-ethereum. https://geth.ethereum.org/ (2023)
12. Heimbach, L., Wattenhofer, R.: Eliminating sandwich attacks with the help of game theory. In: Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security. pp. 153–167 (2022)
13. Lehar, A., Parlour, C.A.: Decentralized exchanges. Tech. rep., working paper, University of Calgary and University of California, Berkeley (2021)
14. Lehar, A., Parlour, C.A.: Battle of the bots: Flash loans, miner extractable value and efficient settlement. Miner Extractable Value and Efficient Settlement (March 8, 2023) (2023)
15. Liu, X., Chen, W., Zhu, K.: Token incentives and platform competition: A tale of two swaps. Available at SSRN 4176638 (2022)
16. Metamask: Metamask. https://metamask.io/ (2023)
17. Park, A.: The conceptual flaws of constant product automated market making. Available at SSRN 3805750 (2021)
18. Qin, K., Zhou, L., Gervais, A.: Quantifying blockchain extractable value: How dark is the forest? In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 198–214. IEEE (2022)
19. Sushiswap: Sushiswap. https://www.sushi.com/ (2021)
20. Thaler, R.H., Benartzi, S.: Save more tomorrow™: Using behavioral economics to increase employee saving. Journal of political Economy **112**(S1), S164–S187 (2004)
21. Thaler, R.H., Sunstein, C.R.: Nudge: improving decisions about health. Wealth, and Happiness **6**, 14–38 (2008)
22. Tokenlon: Tokenlon library. https://tokenlon.gitbook.io/docs/v/docs.en/ (2022)
23. Torres, C.F., Camino, R., et al.: Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 1343–1359 (2021)
24. Uniswap: Uniswap. https://uniswap.org (2021)

25. Uniswap: Uniswap interface. https://github.com/Uniswap/interface/blob/main/src/hooks/useAutoSlippageTolerance.ts\#L75 (2023)
26. Uniswap: Uniswap interface - dynamic slippage. https://github.com/Uniswap/interface/commit/0923cf4ac977f321b4cf11295f7d95b01beae91a (2023)
27. Wang, T., Zhao, C., Yang, Q., Zhang, S., Liew, S.C.: Ethna: Analyzing the underlying peer-to-peer network of ethereum blockchain. IEEE Transactions on Network Science and Engineering **8**(3), 2131–2146 (2021). https://doi.org/10.1109/TNSE.2021.3078181
28. Xu, J., Paruch, K., Cousaert, S., Feng, Y.: SoK: Decentralized Exchanges (DEX) with Automated Market Maker (AMM) Protocols. arXiv preprint arXiv:2103.12732 (2023)
29. Yaish, A., Dotan, M., Qin, K., Zohar, A., Gervais, A.: Suboptimality in defi. Cryptology ePrint Archive (2023)
30. Zhou, L., Qin, K., Torres, C.F., Le, D.V., Gervais, A.: High-frequency trading on decentralized on-chain exchanges. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 428–445. IEEE (2021)
31. Züst, P.: Analyzing and preventing sandwich attacks in ethereum (2021)

A Appendix

A.1 Detection of Sandwich Attacks on Decentralized Exchanges

In this paper, we address the issue of sandwich attacks in liquidity pools by presenting a straightforward measurement approach to identify and detect such attacks using trading data from DEX platforms. Our approach builds upon previous research, including studies by Lehar and Parlour [14], Züst [31], Qin et al. [18] and Torres et al [23].

Specifically, we use the following rules for identifying attacks:

(A) At least two swaps T_{M1} and T_{M2} are included in the same block and swap assets in the same liquidity pool.
(B) T_{M1} and T_{M2} have different transaction hashes.
(C) T_{M1} and T_{M2} swap in different directions.

(D_1) T_{M1} and T_{M2} are initiated by the same wallet.
(D_2) Input amount of T_{M1} equals output amount of T_{M2}, or output amount of T_{M1} equals input amount of T_{M2}.

Additionally, we have the following rules for identifying successful attacks:

(E) At least one additional swap T_A is included in the same block that swapped assets in the same liquidity pool.
(F) T_A executes between T_{M1} and T_{M2}.
(G) T_A and T_{M1} swap in the same direction.
(H) If more than one swap executes between T_{M1} and T_{M2}, then T_{M1} should swap in the same direction as all additional swaps, T_A, T_B, etc.

Rule (A) considers only attacks that are fully executed within the same block. It should be noted that a sandwich attack can still be successful even if T_{M1} and T_{M2} are located in different blocks. However, attackers prefer to have T_{M1} and T_{M2} placed in the same block to minimize the risk of compromising their profits and allowing other users to benefit from them.

Some transactions perform two swaps on the same liquidity pool but in opposing directions. [10] We adopt rule (B) from [23], which ensures that we do not identify such transactions as a sandwich attack.

The contributions of our method can be summarized into two primary aspects. Firstly, we expand the detection capabilities of sandwich attacks to encompass multi-meat attacks, wherein a single attack targets multiple victims simultaneously. Such attacks involve the inclusion of at least two victims (referred to as layers/meat slices in the sandwich) within a single sandwich attack. To illustrate, a sandwich attack involving three victims is commonly referred to as a multi-meat attack with three layers. Conversely, a sandwich attack involving only one layer is commonly referred to as a single-meat attack. Notably, we observed a total of more than 6,000 instances of multi-meat attacks (6.3% of the successful attacks) in Uniswap and Sushiswap in March (Table 6).

[10] Rule (B) removes the noise of misidentified attacks due to the use of aggregator apps which collect several users' swaps into the same transaction. For example, Tokenlon [22] does such an aggregation. For these apps, it is possible for our rules erroneously classify transactions as attacks when two users intend to swap in opposite directions using the same pool.

Secondly, we introduce more flexible rules for analyzing attackers' decisions regarding "cash back," thereby facilitating enhanced detection of a wider range of attacks. The concept of "cash back" refers to the choice of tokens in which the attacker wishes to retain their profits (Token A or B). Previous studies have employed heuristics that restrict the attacker from retaining their profit exclusively in Token B or Token A.

Meat Layer	Attacks	Percentage
1	102,248	93.7%
2	5,116	4.7%
3	1,100	1.0%
≥4	156	0.6%

Table 6: Layers of Sandwich Attack.

For example, in Figure 6, the malicious attacker decides to withdraw 0.2 Tokens B as their profit. However, in real-world scenarios, attackers may have liquidity constraints or specific preferences for other tokens, which can influence their decision-making process. To address this, we have introduced more flexible rules for the "cash back" decision, allowing attackers to choose any combination of tokens.

In previous studies, only (D_2), or modified versions of (D_2), are used. In our study, (D_1) and (D_2) do not need to be satisfied at the same time: only one of the two needs to be satisfied. By adding (D_1), we are able to address the "cash back" issue as mentioned above.

Out of the total 109,120 successful attacks we detected, most of the attacks satisfied (D_1) and (D_2) at the same time. However, by adding Rule (D_1), we detect more attacks than by just using Rule (D_2) alone (see Table 7).

Satisfy D_1	Satisfy D_2	Number of successful attacks	Percentage of successful attacks
Yes	Yes	72,624	66.6%
Yes	No	33,989	31.1%
No	Yes	2,507	2.3%

Table 7: Number of attacks satisfy Rule D_1 and D_2.

Block Size	Number of successful attacks
3 (Rule I)	78,885
4	18,552
≥5	11,683

Table 8: Expansion from common heuristics.

Some other examples of common heuristics used in other works include:

(E)′ Exactly one swap T_A is executed between T_{M1} and T_{M2}.
(I) Consider blocks that contain exactly three transactions.
(J) T_{M1} is the first executed transaction for the traded liquidity pool in a block.

Rule (I) will omit almost 30k successful attacks (see Table 8).

A.2 Swap Data

Uniswap	Number of Swaps	Total value traded ($)	Number of Unique Pools
All data (March 1st to March 31st)	4,612,958 (100%)		17,687 (100%)
Filtered data (Pools with top 1000 tokens)	2,253,712 (48.9%)		2153 (12.2%)
Filtered data (Pools with top 1000 tokens) (Also appear everyday)	2,096,498 (45.4%)	5.32×10^{10}	721 (4.1%)
Sushiswap	Number of Swaps	Total value traded ($)	Number of Unique Pools
All data (March 1st to March 31st)	219,412 (100%)		1,190 (100%)
Filtered data (Pools with top 1000 tokens)	184,686 (84.2%)		401 (33.7%)
Filtered data (Pools with top 1000 tokens) (Also appear everyday)	167,358 (76.3%)	6.64×10^{8}	103 (8.7%)

Table 9: Swap Data Overview.

A.3 Additional plots

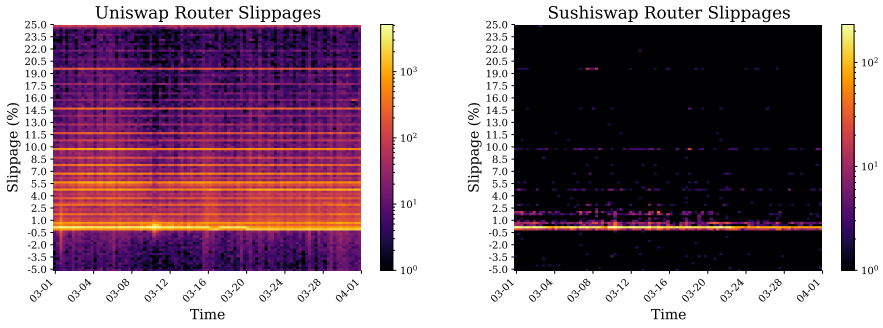

Fig. 5: Inferred router slippages over time, larger range. Color is drawn on a log scale to show detail. Each colored cell counts the number of swaps in a slippage range at a segment of time.

A.4 Sandwich attack

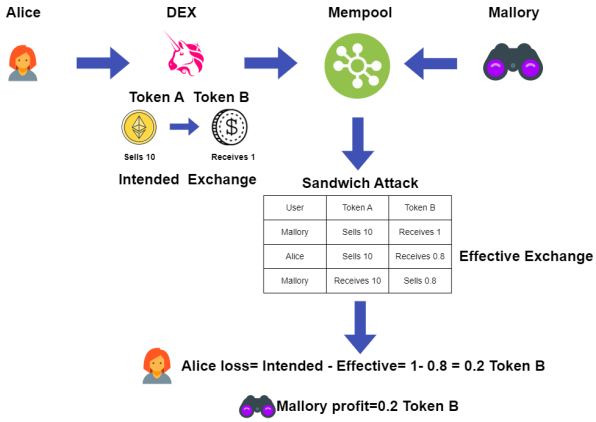

Fig. 6: Sandwich attack example: the transactions introduced by Mallory before and after Alice's transaction result in a worse exchange for Alice and a profit for Mallory.

Depicting a scenario where Mallory observes transactions of both Alice and Bob in the mempool, thereby enabling the execution of a multi-meat attack as shown in Figure 7. Such attacks involve the inclusion of two or more victims (referred to as layers/meat slices in the sandwich) within a single sandwich attack.

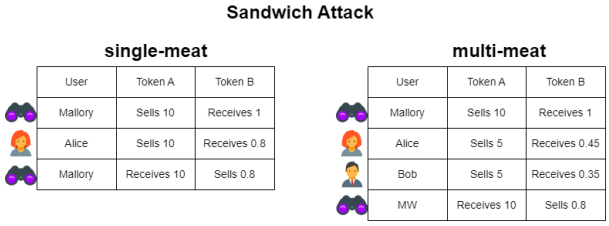

Fig. 7: Single versus multi-meat attacks.