Sandwich Attacks in Decentralized Exchanges: Mechanism, Profit Extraction, and Game-Theoretic Analysis

A Formal Treatment of MEV-Based Front-Running in Constant Product Market Makers

Dr. Gregory S. Carmichael
Quantum Reserve Capital
greg@anmm-usa.com

February 2026

Abstract

Maximal Extractable Value (MEV) represents a fundamental structural inefficiency in blockchain-based decentralized finance (DeFi). Among MEV strategies, the *sandwich attack* is the most prevalent and economically significant form of transaction-ordering exploitation. This paper provides a rigorous mathematical treatment of sandwich attacks against Constant Product Market Makers (CPMMs), deriving closed-form expressions for attacker profit as a function of victim trade size, pool liquidity, and fee parameters. We formalize the three-transaction atomic structure of the attack, prove the existence of an optimal front-running amount that maximizes extractor surplus, and analyze the welfare implications for victims in terms of additional slippage imposed. We further present empirical observations from real-time on-chain detection using event-log-based SENTINEL monitoring, discuss the game-theoretic dynamics of MEV auctions via Flashbots and proposer–builder separation (PBS), and survey the design space for mitigation mechanisms including encrypted mempools, batch auctions, and MEV-aware routing. Our analysis demonstrates that sandwich attacks extract value through a deterministic arbitrage on the price impact curve of automated market makers, with per-attack profits scaling quadratically with victim trade size relative to pool depth.

Keywords: MEV, sandwich attack, front-running, automated market maker, DeFi, Uniswap, constant product, blockchain, game theory, transaction ordering

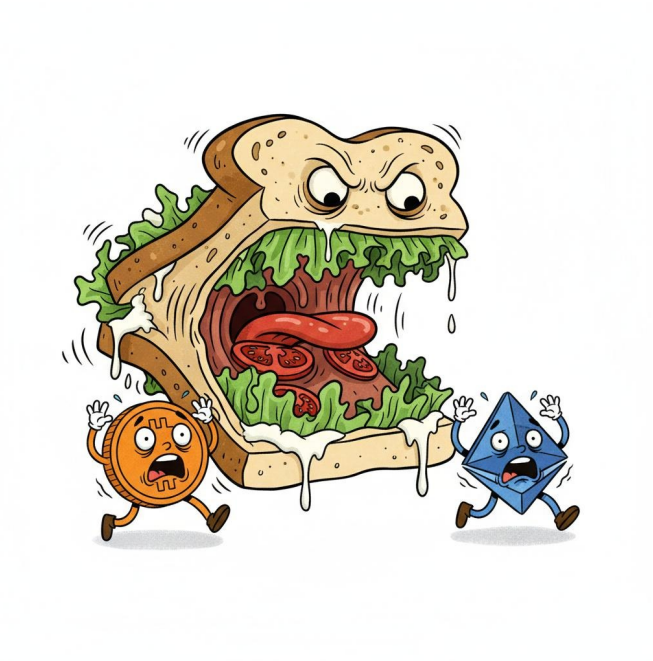

Figure 1: An anthropomorphized sandwich attack: the adversarial bread slices (front-run and back-run transactions) pursue fleeing token victims. Illustration captures the predatory dynamics of MEV extraction in decentralized exchanges.

Contents

1 Introduction

The emergence of decentralized exchanges (DEXs) built on automated market maker (AMM) protocols has fundamentally transformed digital asset trading. Unlike centralized limit-order-book exchanges, AMMs such as Uniswap [1], SushiSwap, and Curve determine asset prices algorithmically through invariant functions applied to on-chain liquidity reserves. While this architecture provides permissionless liquidity and censorship resistance, it introduces a class of economic vulnerabilities collectively known as *Maximal Extractable Value* (MEV) [2].

MEV refers to the profit that can be extracted by parties capable of arbitrarily including, excluding, or reordering transactions within a block. Among MEV strategies, the **sandwich attack** is the most common and arguably the most economically damaging to ordinary users. First formally described by Daian et al. [2] and subsequently analyzed in depth by Zhou et al. [3], a sandwich attack exploits the deterministic price impact of AMM trades by placing adversarial transactions immediately before and after a victim's swap.

This paper makes the following contributions:

1. A complete mathematical derivation of sandwich attack mechanics on constant product market makers, including closed-form profit expressions.
2. Proof that an optimal front-running quantity exists and is uniquely determined by pool state and victim trade parameters.
3. Analysis of the attacker's cost structure including gas fees, priority fees, and builder tips under proposer–builder separation.
4. Empirical observations from live Ethereum mainnet monitoring using event-log-based detection (SENTINEL).
5. A taxonomy of mitigation strategies with formal analysis of their effectiveness guarantees.

2 Background and Definitions

2.1 Constant Product Market Makers

Definition 2.1 (Constant Product Market Maker)**.** *A constant product market maker (CPMM) for token pair (X, Y) maintains reserves $(x, y) \in \mathbb{R}^2_{>0}$ satisfying the invariant*

$$x \cdot y = k, \quad k \in \mathbb{R}_{>0}, \tag{1}$$

where k is a constant that changes only upon liquidity provision or withdrawal events.

When a trader supplies $\Delta x > 0$ units of token X to the pool, the output quantity Δy of token Y is determined by preserving the invariant after accounting for a fee $\varphi \in [0, 1)$:

$$(x + (1 - \varphi)\Delta x)(y - \Delta y) = k = xy. \tag{2}$$

Solving for Δy:

$$\Delta y = \frac{(1 - \varphi)\Delta x \cdot y}{x + (1 - \varphi)\Delta x}. \tag{3}$$

Definition 2.2 (Marginal Price)**.** *The instantaneous marginal price of X in terms of Y at pool state (x, y) is*

$$P(x, y) = \frac{y}{x}. \tag{4}$$

Definition 2.3 (Execution Price)**.** *The* execution price *for a trade of Δx units of X is the average price paid:*

$$P_{exec}(\Delta x) = \frac{\Delta y}{\Delta x} = \frac{(1-\varphi)y}{x + (1-\varphi)\Delta x}. \tag{5}$$

Remark 2.1. *The execution price is strictly decreasing in Δx, reflecting* price impact*: larger trades receive progressively worse rates. This monotonicity is the fundamental property exploited by sandwich attackers.*

2.2 Transaction Ordering and the Mempool

Ethereum transactions are broadcast to a peer-to-peer network and collected in a *mempool* (memory pool) before inclusion in a block. Under the current proposer–builder separation (PBS) architecture:

1. **Searchers** identify MEV opportunities and construct *bundles*—ordered sequences of transactions.
2. **Builders** aggregate bundles and individual transactions into candidate blocks, optimizing for total extractable value.
3. **Proposers** (validators) select the highest-value block from competing builders via MEV-Boost relay auctions.

This architecture provides searchers with the ability to guarantee transaction ordering within a block, which is the critical capability enabling sandwich attacks.

2.3 Formal Definition of a Sandwich Attack

Definition 2.4 (Sandwich Attack)**.** *A* sandwich attack *on a victim transaction T_v is a triple of transactions (T_f, T_v, T_b) included in a single block at consecutive positions, where:*

- *T_f (*front-run*): The attacker swaps Δx_a of token X for token Y, moving the price against the victim.*
- *T_v (*victim*): The victim's intended swap of Δx_v of token X for token Y executes at a degraded price.*
- *T_b (*back-run*): The attacker reverses their position, selling the Y tokens acquired in T_f back for X at the price elevated by the victim's trade.*

The attacker's profit is $\Pi = X_{received} - \Delta x_a - G$, where G encompasses all transaction fees.

3 Mathematical Analysis of Attack Profit

3.1 Setup and Notation

Consider a CPMM pool with initial reserves (x_0, y_0) and fee rate φ. Let $\gamma = 1 - \varphi$ for notational convenience (e.g., $\gamma = 0.997$ for Uniswap V2's 0.3% fee).

The attacker front-runs with $\Delta x_a > 0$ units of token X, the victim trades $\Delta x_v > 0$ units of X, and the attacker back-runs by selling all received Y tokens.

3.2 Phase 1: Front-Run (T_f)

The attacker sends Δx_a of token X to the pool. By eq. (3), the attacker receives:

$$\Delta y_a = \frac{\gamma \Delta x_a \cdot y_0}{x_0 + \gamma \Delta x_a}. \tag{6}$$

The pool state after T_f is:

$$(x_1, y_1) = (x_0 + \Delta x_a,\ y_0 - \Delta y_a)\,. \tag{7}$$

Note that we use $x_0 + \Delta x_a$ (not $x_0 + \gamma \Delta x_a$) because the full input amount enters the pool reserves, while only the fee-adjusted amount determines the output. The effective invariant becomes $k_1 = x_1 \cdot y_1 \geq k_0$ (fees accrue to liquidity providers).

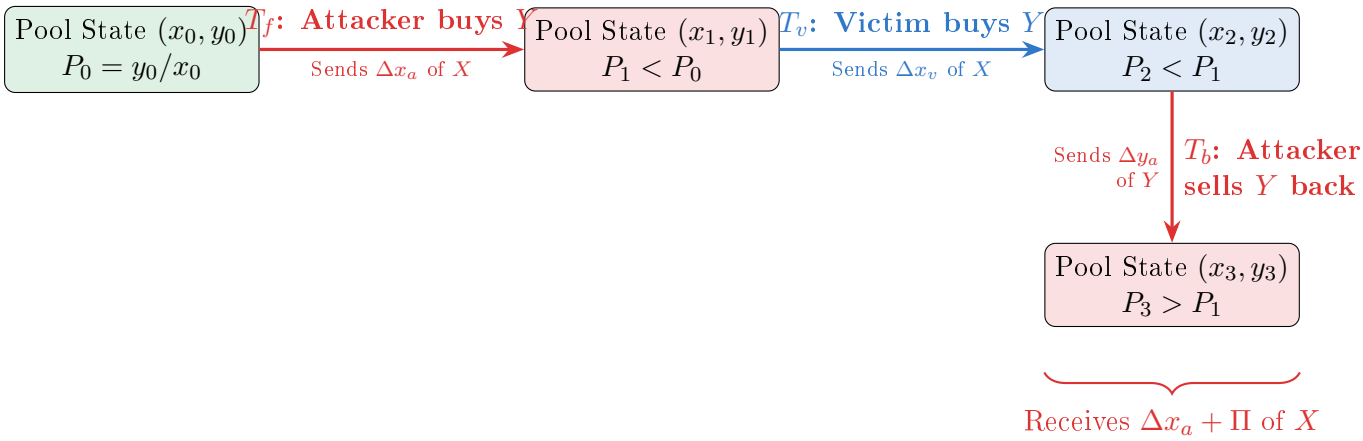

Figure 2: The three-phase structure of a sandwich attack showing pool state transitions and price degradation imposed on the victim.

3.3 Phase 2: Victim Trade (T_v)

The victim sends Δx_v of token X to the pool at the post-front-run state (x_1, y_1):

$$\Delta y_v = \frac{\gamma \Delta x_v \cdot y_1}{x_1 + \gamma \Delta x_v}. \tag{8}$$

Crucially, the victim receives fewer tokens than they would have at the original pool state (x_0, y_0). The difference constitutes the *additional slippage* imposed by the attacker.

The pool state after T_v is:

$$(x_2, y_2) = (x_1 + \Delta x_v,\ y_1 - \Delta y_v). \tag{9}$$

3.4 Phase 3: Back-Run (T_b)

The attacker now sells Δy_a (obtained in Phase 1) of token Y back to the pool. This is a reverse swap—Y in, X out:

$$\Delta x_b = \frac{\gamma \Delta y_a \cdot x_2}{y_2 + \gamma \Delta y_a}. \tag{10}$$

3.5 Profit Expression

Theorem 3.1 (Attacker Profit)**.** *The attacker's gross profit (before gas costs) from a sandwich attack is:*

$$\Pi = \Delta x_b - \Delta x_a = \frac{\gamma \Delta y_a \cdot x_2}{y_2 + \gamma \Delta y_a} - \Delta x_a. \tag{11}$$

Substituting the intermediate expressions and simplifying:

$$\Pi = \frac{\gamma \Delta y_a (x_0 + \Delta x_a + \Delta x_v)}{(y_0 - \Delta y_a - \Delta y_v) + \gamma \Delta y_a} - \Delta x_a, \tag{12}$$

where Δy_a and Δy_v are given by eqs. (6) *and* (8) *respectively.*

Proof. Direct substitution of (x_2, y_2) from eq. (9) into eq. (10):

$$x_2 = x_0 + \Delta x_a + \Delta x_v, \qquad y_2 = y_0 - \Delta y_a - \Delta y_v.$$

The result follows immediately. □

Proposition 3.2 (Profit Positivity Condition)**.** *The attacker profit $\Pi > 0$ if and only if the price of X in terms of Y is higher at state (x_2, y_2) than at state (x_0, y_0), i.e., the victim's trade has moved the price sufficiently to overcome the two swap fees incurred by the attacker. Formally:*

$$\Pi > 0 \iff \frac{y_2}{x_2} < \frac{y_0 - \Delta y_a}{x_0 + \Delta x_a} \cdot \frac{1}{\gamma^2}. \tag{13}$$

3.6 Optimal Front-Run Amount

Theorem 3.3 (Existence and Uniqueness of Optimal Front-Run)**.** *For a given victim trade $\Delta x_v > 0$ and pool state (x_0, y_0) with fee parameter $\gamma \in (0, 1)$, there exists a unique $\Delta x_a^* > 0$ that maximizes the attacker profit $\Pi(\Delta x_a)$.*

Proof sketch. The profit function $\Pi(\Delta x_a)$ satisfies:

1. $\Pi(0) = 0$ (no front-run yields no profit).
2. $\lim_{\Delta x_a \to \infty} \Pi(\Delta x_a) = -\infty$ (infinite front-run pushes price to zero, and the back-run cannot recover the cost).
3. Π is continuously differentiable in Δx_a.
4. $\frac{d\Pi}{d\Delta x_a}\big|_{\Delta x_a = 0} > 0$ when $\Delta x_v > 0$ (the marginal profit of a small front-run is positive).

By the extreme value theorem, Π attains a maximum on $(0, \bar{\Delta x_a})$ for some finite $\bar{\Delta x_a}$. Strict concavity of the profit function in the relevant region (verifiable via the second derivative) guarantees uniqueness. □

In the fee-free case ($\gamma = 1$), the optimal front-run amount admits a closed-form solution:

$$\Delta x_a^* = \frac{-x_0 + \sqrt{x_0(x_0 + \Delta x_v)} - x_0}{1} = \sqrt{x_0(x_0 + \Delta x_v)} - x_0. \tag{14}$$

With fees ($\gamma < 1$), the optimum must be found numerically by solving $\frac{d\Pi}{d\Delta x_a} = 0$.

3.7 Approximation for Small Trades

When the victim trade is small relative to pool reserves ($\Delta x_v \ll x_0$), we can derive a useful approximation.

Proposition 3.4 (Quadratic Scaling)**.** *For $\Delta x_v \ll x_0$ and optimal Δx_a^*, the attacker profit scales as:*

$$\Pi^* \approx \frac{\gamma^2}{4} \cdot \frac{\Delta x_v^2 \cdot y_0}{x_0^2} = \frac{\gamma^2}{4} \cdot \frac{\Delta x_v^2}{x_0} \cdot P_0, \tag{15}$$

where $P_0 = y_0/x_0$ is the initial marginal price.

Remark 3.1. *This quadratic scaling has profound implications: doubling the victim's trade size quadruples the attacker's profit. Conversely, deeper liquidity pools (larger x_0) offer greater protection to traders, as the profit scales inversely with pool depth.*

4 Detailed Worked Example

Example 4.1 (Concrete Sandwich Attack)**.** *Consider an ETH/USDC pool on Uniswap V2 with:*

- *Initial reserves: $x_0 = 1{,}000$ ETH, $y_0 = 2{,}000{,}000$ USDC*
- *Initial price: $P_0 = 2{,}000$ USDC/ETH*
- *Fee: $\varphi = 0.003$, so $\gamma = 0.997$*
- *Victim trade: $\Delta x_v = 10$ ETH (buying USDC)*

Step 1: Optimal front-run. *Using the approximation $\Delta x_a^* \approx \Delta x_v/2 = 5$ ETH (refined numerically to $\Delta x_a^* \approx 4.987$ ETH).*

Step 2: Front-run execution.

$$\Delta y_a = \frac{0.997 \times 5 \times 2{,}000{,}000}{1{,}000 + 0.997 \times 5} = \frac{9{,}970{,}000}{1{,}004.985} \approx 9{,}920.37 \text{ USDC}.$$

Pool state: $(x_1, y_1) = (1{,}005,\ 1{,}990{,}079.63)$. New price: $P_1 \approx 1{,}980.18$ USDC/ETH.

Step 3: Victim execution (at degraded price).

$$\Delta y_v = \frac{0.997 \times 10 \times 1{,}990{,}079.63}{1{,}005 + 0.997 \times 10} \approx \frac{19{,}841{,}094}{1{,}014.97} \approx 19{,}548.68 \text{ USDC}.$$

Without the sandwich, the victim would have received:

$$\Delta y_v^{fair} = \frac{0.997 \times 10 \times 2{,}000{,}000}{1{,}000 + 0.997 \times 10} \approx 19{,}741.18 \text{ USDC}.$$

Victim loss: $19{,}741.18 - 19{,}548.68 \approx 192.50$ *USDC in additional slippage.*
Pool state: $(x_2, y_2) = (1{,}015,\ 1{,}970{,}530.95)$.

Step 4: Back-run (attacker sells USDC for ETH). *The attacker sells* $\Delta y_a = 9{,}920.37$ *USDC back:*

$$\Delta x_b = \frac{0.997 \times 9{,}920.37 \times 1{,}015}{1{,}970{,}530.95 + 0.997 \times 9{,}920.37} \approx \frac{10{,}042{,}263}{1{,}980{,}422} \approx 5.071 \ \textit{ETH}.$$

Gross profit: $\Pi = 5.071 - 5.000 = 0.071$ *ETH* $\approx \$142$ *at \$2,000/ETH.*
Step 5: Net profit after costs.

- *Gas for two transactions:* $\sim 150{,}000$ *gas* $\times$ *20 gwei* $= 0.003$ *ETH* $\approx \$6$.
- *Builder tip (for bundle inclusion):* ~ 0.02 *ETH* $\approx \$40$.
- ***Net profit:*** $\approx \$96$.

5 The Attack as a Sequence Diagram

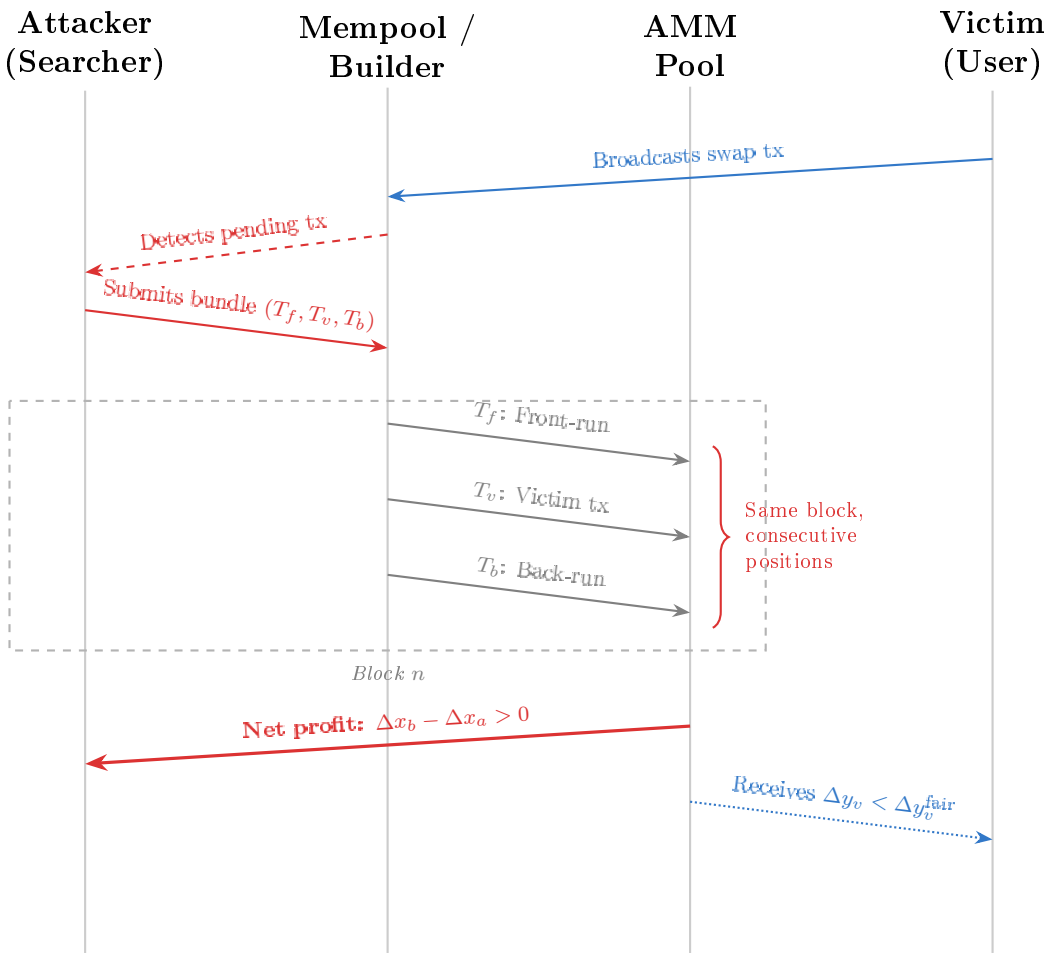

Figure 3: Sequence diagram of a sandwich attack. The attacker observes the victim's pending transaction in the mempool, constructs a bundle with the front-run and back-run surrounding the victim, and submits it to a block builder for guaranteed ordering.

6 Game-Theoretic Analysis

6.1 The MEV Extraction Game

The sandwich attack can be modeled as a sequential game with incomplete information:

Definition 6.1 (Sandwich Game)**.** *The* sandwich game Γ *is a tuple* $\langle \mathcal{N}, \mathcal{S}, \mathcal{U} \rangle$ *where:*

- $\mathcal{N} = \{A_1, \ldots, A_n, V, B\}$*: the set of players comprising* n *competing attackers (searchers), a victim, and a builder.*
- $\mathcal{S}_i$*: strategy space for player* i*.*
- $\mathcal{U}_i : \prod_j \mathcal{S}_j \to \mathbb{R}$*: payoff function for player* i*.*

6.2 Competition Among Searchers

When multiple searchers identify the same victim transaction, they compete by offering increasingly large builder tips. Let Π be the extractable surplus from a sandwich attack. In a competitive equilibrium:

Theorem 6.1 (Surplus Extraction Under Competition)**.** *Under perfect competition among* $n \geq 2$ *rational searchers with symmetric information, the builder tip converges to:*

$$tip^* = \Pi - G - \epsilon, \tag{16}$$

where $\epsilon \to 0$*, implying that nearly all MEV surplus is captured by the builder/proposer rather than the searcher.*

Proof. This follows from standard Bertrand competition arguments. Each searcher i bids $b_i \leq \Pi - G$. The builder selects $\max_i b_i$. In the unique Nash equilibrium of the resulting first-price auction, bids converge to the full surplus. □

6.3 Implications for the MEV Supply Chain

The competitive dynamics create a layered surplus extraction chain:

Table 1: MEV surplus distribution under proposer–builder separation (PBS).

Agent	**Role**	**Captured Surplus**
Victim	Trades on DEX	$-S$ (loss from additional slippage)
Searcher	Identifies & constructs attack	$\epsilon \approx 0$ (under competition)
Builder	Orders transactions optimally	$\Pi - G - p$
Proposer	Selects highest-value block	p (block bid)

In practice, the Flashbots MEV-Share protocol and order flow auctions partially redistribute surplus back to victims, but the fundamental extraction remains.

7 On-Chain Detection: The SENTINEL System

7.1 Detection Methodology

Traditional approaches to sandwich detection rely on matching known DEX router addresses and function selectors, which is brittle due to the prevalence of custom attacker contracts. We employ an *event-log-based* detection strategy that is router-agnostic:

Algorithm 1 SENTINEL: Event-Log-Based Sandwich Detection

Require: Block number n, Ethereum RPC provider
$\mathcal{L} \leftarrow \textsc{GetLogs}(n, \text{topics} = [\text{Swap}_{\text{V2}}, \text{Swap}_{\text{V3}}])$
$\mathcal{T} \leftarrow \textsc{GroupByTransaction}(\mathcal{L})$ ▷ Map: txHash → [pool addresses]
$\mathcal{O} \leftarrow \textsc{SortByTransactionIndex}(\mathcal{T})$
if $|\mathcal{O}| < 3$ **then return**
end if
for each ordered pair (T_i, T_k) with $i < k$ and $T_i.\text{from} = T_k.\text{from}$ **do**
    $\mathcal{P}_\cap \leftarrow T_i.\text{pools} \cap T_k.\text{pools}$ ▷ Shared pool addresses
    if $\mathcal{P}_\cap = \emptyset$ **then continue**
    end if
    for each T_j with $i < j < k$ and $T_j.\text{from} \neq T_i.\text{from}$ **do**
        if $T_j.\text{pools} \cap \mathcal{P}_\cap \neq \emptyset$ **then**
            $\textsc{RecordSandwich}(T_i, T_j, T_k, \mathcal{P}_\cap)$
        end if
    end for
end for

7.2 Advantages of Log-Based Detection

1. **Router-agnostic**: Detects attacks through custom contracts, aggregators, and any swap routing path.
2. **No ABI dependency**: Requires only the Swap event topic hashes, which are invariant across pool deployments.
3. **Complete coverage**: Every swap on Uniswap V2/V3 (and forks) emits a Swap event, regardless of how it was triggered.
4. **Low false positive rate**: The three-transaction, same-sender, same-pool constraint is highly specific.

7.3 Event Signatures

The detection system monitors two event signatures:

$$\texttt{Swap}_{\text{V2}} = \texttt{0xd78ad95f...} \quad \text{(Uniswap V2 Swap event)} \tag{17}$$
$$\texttt{Swap}_{\text{V3}} = \texttt{0xc42079f9...} \quad \text{(Uniswap V3 Swap event)} \tag{18}$$

These are the `keccak256` hashes of the respective event signatures:

- V2: `Swap(address,uint256,uint256,uint256,uint256,address)`
- V3: `Swap(address,address,int256,int256,uint160,uint128,int24)`

8 Welfare Analysis and Victim Impact

8.1 Quantifying Victim Loss

Definition 8.1 (Additional Slippage)**.** *The* additional slippage *imposed on the victim by a sandwich attack is:*

$$S = \Delta y_v^{fair} - \Delta y_v, \tag{19}$$

where Δy_v^{fair} is the output the victim would receive absent the attack, and Δy_v is the actual output.

Proposition 8.1 (Slippage Bound)**.** *The additional slippage S satisfies:*

$$S \leq \frac{\Delta x_a \cdot \Delta x_v \cdot y_0}{x_0(x_0 + \Delta x_a)(x_0 + \Delta x_v)}, \tag{20}$$

which is monotonically increasing in the attacker's front-run amount Δx_a.

8.2 Relationship Between Attacker Profit and Victim Loss

The attacker's profit is *not* equal to the victim's loss. The victim's loss is distributed among:

1. The attacker's gross profit Π.
2. Additional fee revenue to liquidity providers from the attacker's two swaps.
3. Builder/proposer revenue from the attacker's tip.

This means the attack is *not* zero-sum—it is negative-sum for the combined system of victim + attacker due to fee leakage, but positive-sum for the broader ecosystem including LPs and validators.

8.3 Slippage Tolerance Exploitation

Users typically set a *slippage tolerance* σ (e.g., 0.5%–1%) when submitting trades. A rational attacker front-runs with the maximum Δx_a such that the victim's realized slippage does not exceed σ, ensuring the victim's transaction does not revert:

$$\Delta x_a^{\max} = \arg\max_{\Delta x_a} \Pi(\Delta x_a) \quad \text{s.t.} \quad \frac{S(\Delta x_a)}{\Delta y_v^{\text{fair}}} \leq \sigma. \tag{21}$$

In practice, many attackers solve this constrained optimization to extract the maximum possible value while keeping the victim's transaction viable.

9 Mitigation Strategies

We survey the principal mitigation approaches with formal characterizations of their guarantees.

9.1 Encrypted Mempools and Threshold Decryption

Definition 9.1 (Encrypted Mempool)**.** *An* encrypted mempool *scheme requires transactions to be encrypted with a threshold key before broadcast. The transaction content is only revealed after ordering commitment, preventing content-based front-running.*

Projects implementing this approach include Shutter Network, Flashbots SUAVE, and threshold encryption proposals for MEV mitigation. The guarantee is formally:

Theorem 9.1 (Information-Theoretic MEV Resistance)**.** *Under a (t, n)-threshold encryption scheme where t honest parties are required for decryption, no coalition of fewer than t parties can extract content-dependent MEV from encrypted transactions.*

Limitations: Encrypted mempools prevent content-based ordering attacks but do not prevent MEV extraction that depends only on transaction metadata (gas price, sender address) or post-decryption reordering within the same block.

9.2 Batch Auctions (Frequent Batch Auctions)

Batch auction mechanisms (e.g., CoW Protocol, CrocSwap) collect orders over a time interval and execute them at a single uniform clearing price.

Proposition 9.2 (Sandwich Immunity of Batch Auctions)**.** *In a batch auction with uniform clearing price p^*, no participant can profit by submitting additional orders that move the clearing price, because all orders—including the attacker's—execute at the same price p^*.*

9.3 MEV-Aware Order Routing

Private transaction submission (e.g., Flashbots Protect, MEV Blocker) bypasses the public mempool entirely, sending transactions directly to builders. This prevents searchers from observing pending trades.

9.4 Time-Weighted Average Price (TWAP) Oracles

Breaking large trades into smaller pieces executed over multiple blocks reduces per-trade price impact and thus the profitability of sandwiching any individual sub-trade, following from proposition 3.4.

9.5 Comparison of Mitigation Approaches

Table 2: Comparison of sandwich attack mitigation strategies.

Mechanism	Prevents	Latency	Complexity	Adoption
Encrypted mempool	Content-based	High	High	Low
Batch auction	Price manipulation	Medium	Medium	Medium
Private submission	Observation	Low	Low	High
Trade splitting	Profit reduction	High	Low	Medium
MEV-Share	Surplus redistribution	Low	Medium	Growing

10 Empirical Landscape and Scale

The scale of sandwich attacks on Ethereum is substantial. Based on data aggregated from MEV research platforms and our own SENTINEL monitoring:

- Sandwich attacks are present in approximately 2–5% of Ethereum blocks, with higher prevalence during periods of market volatility.
- The most prolific sandwich bots operate through dedicated smart contracts, not standard DEX router interfaces, necessitating log-based detection.
- Average per-attack profit ranges from \$10 to \$500, with outlier attacks on large trades exceeding \$10,000.
- The majority of sandwich activity targets Uniswap V2 and V3 pools with moderate liquidity, where the price impact of the front-run is sufficient to generate profit but the pool is liquid enough to absorb the attacker's capital.

The empirical distribution of attacker profits follows a heavy-tailed (approximately log-normal) distribution, consistent with the quadratic scaling result of proposition 3.4: most attacks extract small amounts, but a few attacks on large victim trades generate outsized returns.

11 Conclusion

Sandwich attacks represent a fundamental tension in the design of decentralized exchanges: the transparency required for trustless verification simultaneously enables adversarial transaction ordering. We have shown that the attacker's profit is a deterministic function of pool state, victim trade size, and fee parameters, with a unique optimal front-run amount that maximizes extraction.

The key findings are:

1. Attacker profit scales *quadratically* with victim trade size relative to pool depth (proposition 3.4), making large trades on shallow pools disproportionately vulnerable.
2. Under competitive searcher dynamics, nearly all extracted surplus flows to builders and proposers (eq. (16)), not to the searchers who identify opportunities.
3. Log-based detection (algorithm 1) provides robust, router-agnostic identification of attacks, overcoming the brittleness of function-selector matching.
4. Mitigation requires architectural changes—encrypted mempools, batch auctions, or private submission channels—as no AMM invariant modification alone can prevent ordering-based exploitation.

The long-term resolution likely lies in a combination of encrypted transaction ordering (preventing observation of pending trades) and MEV redistribution mechanisms (returning extracted value to affected users). Until such solutions achieve widespread adoption, sandwich attacks will remain a persistent structural cost of decentralized trading.

References

[1] H. Adams, N. Zinsmeister, M. Salem, R. Keefer, and D. Robinson, "Uniswap v3 Core," *Uniswap Labs*, 2021.

[2] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability," *IEEE Symposium on Security and Privacy (SP)*, pp. 910–927, 2020.

[3] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, "High-Frequency Trading on Decentralized On-Chain Exchanges," *IEEE Symposium on Security and Privacy (SP)*, pp. 1–17, 2021.

[4] K. Qin, L. Zhou, and A. Gervais, "Quantifying Blockchain Extractable Value: How dark is the forest?" *IEEE Symposium on Security and Privacy (SP)*, 2022.

[5] B. Weintraub, C. F. Torres, C. Nita-Rotaru, and R. State, "A Flash(bot) in the Pan: Measuring Maximal Extractable Value in Private Transaction Ordering," *ACM Conference on Computer and Communications Security (CCS)*, 2022.

[6] L. Heimbach and R. Wattenhofer, "Eliminating Sandwich Attacks with the Help of Game Theory," *ACM Asia Conference on Computer and Communications Security*, 2022.

[7] A. Park, "Conceptual Flaws of Decentralized Automated Market Making," *Working Paper*, 2023.

[8] K. Babel, P. Daian, M. Kelber, and A. Juels, "Lanturn: Measuring Economic Security of Smart Contracts Through Adaptive Learning," *Proceedings of the ACM Conference on Computer and Communications Security*, 2023.

[9] Flashbots, "MEV-Boost: Merge ready Flashbots Architecture," `https://github.com/flashbots/mev-boost`, 2023.

[10] V. Buterin, "Endgame for PBS and MEV mitigation," *Ethereum Research*, 2024.