Novel Methods for Manipulation-Resistant TWAPs in the High-Frequency Compute-Limited Discrete Regime

Greshams Code[1]

[1]Founder, Govex.ai

May 2025

Abstract

Time-Weighted Average Price (TWAP) oracles are used in decentralized finance (DeFi) to gauge a stable price for assets. TWAPs are susceptible to manipulation, particularly in high-frequency compute-limited discrete-time environments such as Solana and Sui. Existing TWAP mechanisms often face trade-offs between manipulation resistance and computational efficiency. This paper introduces novel methods for constructing manipulation-resistant TWAPs tailored for these constrained regimes. We propose two primary mechanisms: firstly, a 'stepping cap' that allows the TWAP's effective observation base to adjust retrospectively over multiple windows after large, capped trades, minimizing write operations and enhancing capital efficiency. Secondly, an 'intra-window dynamic capping' strategy that utilizes the effective TWAP of the preceding window as a robust baseline for applying caps in the current window. Together, these methods aim to reduce the cost and feasibility of manipulation, prevent race conditions during window initialization through efficient 'write-once' principles, and improve the overall resilience of on-chain TWAPs without incurring prohibitive computational overhead. We present the algorithmic details of these methods and argue for their improved security and efficiency characteristics compared to conventional approaches.

1 Introduction

Time-Weighted Average Price (TWAP) oracles are integral to decentralized finance (DeFi), providing a mechanism to smooth the stochastic volatility of asset prices and to establish a fair market reference. They are extensively used in critical applications such as Automated Market Makers (AMMs) like Uniswap V4 for dynamic fee calculations or other advanced pool logic, and in governance systems like MetaDAO to determine winning outcomes in Futarchy proposals

[1]. These high-stakes scenarios present significant incentives for attackers to manipulate prices, potentially for exploiting lending protocols or subverting governance decisions.

We introduce two novel mechanisms to enhance TWAP resilience under these constraints. The first, a 'stepping cap', allows the TWAP's effective observation base to adjust retrospectively over multiple windows following large, capped trades. This minimizes write operations and improves capital efficiency by more accurately reflecting sustained price changes over time. The second, an 'intra-window dynamic capping' strategy, employs the effective TWAP of the preceding window as a robust baseline for applying dynamic price caps within the current window. This approach aims to mitigate the impact of sudden, large price swings that could be manipulative.

2 Background and Related Work

2.1 Time-Weighted Average Prices (TWAPs)

A Time-Weighted Average Price (TWAP) is a financial metric that calculates the average price of an asset over a specified period, where each price is weighted by the duration for which it prevailed. Formally, for a series of n price observations P_i each lasting for a time duration Δt_i within a total window $T = \sum_{i=1}^{n} \Delta t_i$, the TWAP is calculated as:

$$\text{TWAP} = \frac{\sum_{i=1}^{n} (P_i \cdot \Delta t_i)}{\sum_{i=1}^{n} \Delta t_i}$$

2.2 Existing TWAP Mechanisms and Applications

Several TWAP implementations are prominent in DeFi:

- **Uniswap V2** pioneered on-chain TWAPs by maintaining cumulative price sums $price0CumulativeLast, price1CumulativeLast$. These accumulators are updated with $price * time_elapsed$ upon liquidity events. Users query the TWAP by taking two readings and calculating the difference over the time elapsed. This design offers $O(1)$ storage but calculates an arithmetic mean, which can be influenced by extreme price swings over sustained periods.
- **Uniswap V3** improved upon this by using a geometric mean TWAP, accumulating the logarithm of the price (via "ticks"). This makes it more resistant to single-block price manipulation as extreme price changes have a less-than-proportional impact on the geometric mean. It stores observations in a circular buffer, allowing for flexible lookback periods but requiring more storage than V2.
- **Uniswap V4** introduces "hooks," which allow for custom logic to be added to pools, including more advanced oracle designs like truncating

using the geometric mean from tick positions. A truncated oracle caps the per-block price movement that the oracle records, forcing manipulators to sustain efforts over many blocks.

- **MetaDAO Futarchy** utilizes a TWAP where updates are restricted to once per 60-second window, and the price can only move by an absolute capped step per observation triggered by an AMM swap [1]. This design explicitly limits the rate of change to deter manipulation in its governance mechanism.

- **Curve Finance** implements Exponential Moving Average (EMA) oracles, which give more weight to recent prices. While EMAs are computationally efficient ($O(1)$ storage and update), their responsiveness can make them cheaper to manipulate if the averaging period (half-life) is short [2].

These oracles are crucial for lending protocols, derivatives, stablecoins, and governance mechanisms.

2.3 Literature on Manipulation Resistance

Research has focused on enhancing TWAP security:

- **Median-Based Oracles:** Using the median price instead of the mean offers strong resistance to outliers, as an attacker must control over 50% of observations in a window to significantly skew the median. Implementations like Euler Finance's Time-Weighted Median Price using a ring buffer and novel algorithms like Ormer, which uses a piecewise-parabolic formula to estimate the median with constant storage, have been proposed. [3] However, on-chain medians can be storage-intensive or computationally complex if not carefully designed. For markets with infrequent trades, such as governance markets where trades might average 150-second gaps, a simple median of recent trades might not be representative or fair if the window captures too few distinct price points [4].

- **Winsorized/Truncated Oracles:** These methods "cap" or "trim" extreme price movements. For instance, Uniswap V4's truncated oracle hook limits the maximum price change recorded by the oracle in a single block. This forces attackers to sustain manipulation over more blocks, increasing costs. Defining caps based on larger window parameters, rather than solely on per-observation limits, can allow for greater precision and robustness in the cap value, especially in high-frequency environments where individual block/checkpoint times (e.g., Sui's 220ms checkpoints, Solana's 400ms blocks [5, 6]) might be too granular for stable cap setting.

- **Zero-Knowledge (ZK) Proofs:** ZK-Median oracles propose computing medians or other complex statistics off-chain and verifying them on-chain with a ZK proof, reducing on-chain computational load. This is promising but still experimental and may introduce latency.

Our work aims to build upon these insights by proposing mechanisms that are inherently efficient and robust within such high-frequency, compute-limited discrete-time settings, focusing on on-chain logic that minimizes state changes while maximizing manipulation resistance.

3 Problem Formulation and Preliminaries

This section formally defines the operational environment, the nature of manipulation we aim to prevent, the metrics for evaluation, and essential notation.

3.1 Defining Manipulation

In the context of this paper, TWAP manipulation refers to actions taken by an adversary to intentionally cause the reported TWAP of an asset to deviate significantly from its perceived "fair" or "true" market price over a target evaluation period. This is typically achieved by:

- Executing atypically large volume trades to momentarily shift the spot price.
- Strategically timing trades around oracle observation or window boundaries.
- Exploiting the specific mechanics of an oracle's price accumulation or update logic.

The manipulator's goal is often to profit from discrepancies in other DeFi protocols that consume this oracle price (e.g., undercollateralized loans, unfair derivative settlements) or to influence governance outcomes based on oracle readings.

3.2 Evaluation Metrics

We evaluate TWAP mechanisms based on their manipulation resistance and efficiency:

- **Manipulation Resistance:**
  - *Cost of Attack (CoA):* The amount of capital an attacker must expend (e.g., in terms of induced slippage, transaction fees, or capital lockup) to shift the reported TWAP by a target percentage $\delta\%$ for a specified duration T_D. Higher CoA indicates better resistance.
  - *Maximum Price Deviation (ϵ_{max}):* The largest percentage difference between the oracle's reported TWAP and a benchmark fair price (e.g., a volume-weighted average price from high-liquidity centralized exchanges) during a manipulation attempt.
  - *Resilience to Spikes:* The ability of the oracle to either ignore transient, large price spikes or recover quickly to a fair price level once the manipulative pressure ceases.

- **Efficiency:**
  - *Computational Cost:* The gas or compute units required per oracle update (processing a new price observation) and per oracle query (reading the TWAP). Measured in terms of arithmetic operations and complexity.
  - *Storage Cost:* The number of persistent storage slots required by the oracle on-chain.
  - *Latency:* The time lag between a significant, legitimate change in the true market price and its reflection in the reported TWAP. While some lag is inherent in TWAPs, excessive lag can be detrimental.
  - *Scalability:* Should work for a 60 minute market up to a 1 year market

4 Design

Derivation of Formula for Retroactive TWAP Step Cap Updates Over Multiple Windows

Let N_W be the number of new windows available for adjustment (e.g., $w - w_{\text{prev}}$). We assume N_W is a non-negative integer. Let $G_{abs} = |P - B|$ be the absolute gap between the current price P and the base price B. Let Δ_M be the maximum allowed change per step (your $\Delta_{\max}$ or delta cap).

The value contributed by window i (for $i = 1, \ldots, N_W$) is $\min(i \cdot \Delta_M, G_{abs})$. We want to calculate the sum of these values:

$$V_{\text{total}} = \sum_{i=1}^{N_W} \min(i \cdot \Delta_M, G_{abs})$$

To find a closed-form sum for V_{total}, we first determine how many window contributions are dictated by the ramping term $i \cdot \Delta_M$ versus the cap G_{abs}.

Let k_{cap_idx} be the index of the first window i (if it exists within G_{abs}/Δ_M) at which the potential unconstrained adjustment $i \cdot \Delta_M$ would meet or exceed G_{abs}:

$$k_{cap\_idx} = \left\lceil \frac{G_{abs}}{\Delta_M} \right\rceil$$

If $G_{abs} = 0$, then $k_{cap\_idx} = 0$. Note that k_{cap_idx} can be 0 if G_{abs} is 0, or 1 if $0 < G_{abs} \leq \Delta_M$.

The number of initial windows whose contribution $i \cdot \Delta_M$ is strictly less than G_{abs} is k_{ramp_limit}:

$$k_{ramp\_limit} = \max(0, k_{cap\_idx} - 1)$$

This means that for $i = 1, \ldots, k_{ramp\_limit}$, the contribution is $i \cdot \Delta_M$. For all subsequent windows $i > k_{ramp\_limit}$, the contribution will be G_{abs} (because $i \cdot \Delta_M \geq k_{cap\_idx} \cdot \Delta_M \geq G_{abs}$).

Let N_{ramp_terms} be the actual number of windows that contribute to the ramp part of the sum. This is limited by both N_W and k_{ramp_limit}:

$$N_{ramp\_terms} = \min(N_W, k_{ramp\_limit})$$

The total conceptual value, V_{total}, can be expressed as the sum of two components:

Ramp Accumulation (V_{ramp})

This is the sum of $i \cdot \Delta_M$ for the first N_{ramp_terms} windows, as these are the windows where $i \cdot \Delta_M < G_{abs}$ and $i \leq N_W$.

$$V_{\text{ramp}} = \sum_{i=1}^{N_{ramp\_terms}} (i \cdot \Delta_M) = \Delta_M \cdot \frac{N_{ramp\_terms}(N_{ramp\_terms} + 1)}{2}$$

If $N_{ramp\_terms} = 0$ (e.g., if $G_{abs} \leq \Delta_M$ making $k_{ramp\_limit} = 0$, or if $N_W = 0$), then $V_{\text{ramp}} = 0$.

Flat Value Accumulation (V_{flat})

The remaining $(N_W - N_{ramp\_terms})$ windows (if any) each contribute G_{abs}. This is because for these windows, either their index $i > k_{ramp\_limit}$ (meaning $i \cdot \Delta_M \geq G_{abs}$) or all N_W windows fall into this category (if $k_{ramp\_limit} = 0$). The number of windows contributing the flat value is $(N_W - N_{ramp\_terms})$.

$$V_{\text{flat}} = G_{abs} \cdot (N_W - N_{ramp\_terms})$$

The total accumulated conceptual value is the sum of these components:

$$V_{\text{total}} = V_{\text{ramp}} + V_{\text{flat}}$$

Substituting the definitions:

$$V_{\text{total}} = \Delta_M \cdot \frac{N_{ramp\_terms}(N_{ramp\_terms} + 1)}{2} + G_{abs} \cdot (N_W - N_{ramp\_terms})$$

This revised formulation correctly calculates the sum $\sum_{i=1}^{N_W} \min(i \cdot \Delta_M, G_{abs})$ for integer N_W and any non-negative G_{abs} and Δ_M.

4.1 Derivation of Formula for Intra-Step TWAP for Previous Window Base

This outlines the calculation of a custom Time-Weighted Average Price ($TWAP_{custom}$) for the current window, W_k. Live market prices are adjusted relative to the previous window's final TWAP, and these adjusted prices are time-weighted.

Definitions

- W_k: The current trading window.
- $T_{start}(W_k)$: Start time of window W_k.
- $T_{end}(W_k)$: End time of window W_k.
- $T_W(W_k) = T_{end}(W_k) - T_{start}(W_k)$: Total duration of window W_k.
- $TWAP_{ref} = TWAP_{final}(W_{k-1})$: The final TWAP of the previous window, W_{k-1}. This is a fixed reference for W_k.
- $P(t_{obs})$: Live market price observed at time t_{obs} within W_k.
- P'_{obs}: The Adjusted Price derived from $P(t_{obs})$.
- Δ_M: Maximum allowed absolute deviation of $P(t_{obs})$ from $TWAP_{ref}$.
- t_{last_obs}: Timestamp of the previously processed observation within W_k. Initially $T_{start}(W_k)$.
- $\Sigma(P' \cdot \Delta t)$: Running sum of (Adjusted Price $\times$ Time Period). Initialized to 0 at the start of W_k.

Adjusting Observed Prices

For each live market price $P(t_{obs})$ observed at t_{obs} within W_k: The Adjusted Price P'_{obs} is calculated as:

$$P'_{obs} = TWAP_{ref} + \text{cap}\left((P(t_{obs}) - TWAP_{ref}), -\Delta_M, \Delta_M\right)$$

Accumulating Value On-the-Fly

When an Adjusted Price P'_{obs} (derived from $P(t_{obs})$) is determined:

The time period, Δt_{eff}, for which the *previous* adjusted price was effective ends at t_{obs}. The previous adjusted price was active from t_{last_obs} up to the current observation time t_{obs}. Let P'_{last} be the adjusted price corresponding to t_{last_obs} (or $TWAP_{ref}$ if $t_{last\_obs} = T_{start}(W_k)$ and no prior P' exists).

The time period for P'_{last} is:

$$\Delta t_{period} = t_{obs} - t_{last\_obs}$$

The contribution to the sum is:

$$\text{Value Contribution} = P'_{last} \cdot \Delta t_{period}$$

Add this to the running sum:

$$\Sigma(P' \cdot \Delta t) \leftarrow \Sigma(P' \cdot \Delta t) + \text{Value Contribution}$$

Then, update $t_{last\_obs} \leftarrow t_{obs}$, and store P'_{obs} as the new P'_{last} for the next iteration.

Handling the final segment of the window

At $T_{end}(W_k)$, the last active P'_{last} (from the final observation t_{last_obs} within W_k) is applied for the remaining period:

$$\Delta t_{final\_period} = T_{end}(W_k) - t_{last\_obs}$$

$$\text{Final Value Contribution} = P'_{last} \cdot \Delta t_{final\_period}$$

$$\Sigma(P' \cdot \Delta t) \leftarrow \Sigma(P' \cdot \Delta t) + \text{Final Value Contribution}$$

Final $TWAP_{custom}(W_k)$ Calculation

After processing all observations and accounting for the final segment up to $T_{end}(W_k)$, the custom TWAP is:

$$TWAP_{custom}(W_k) = \frac{\Sigma(P' \cdot \Delta t)}{T_W(W_k)}$$

5 Simulation Results

5.1 Retrospective TWAP step

Both bots and humans can be expected to want to move the TWAP accumulation price to the raw AMM price. With a naive TWAP method, trades that move the price by more than the TWAP cap would have to be split into multiple separate trades over multiple windows. This new method ensures bots are not prioritized over human traders. Human users find the time and mental overhead complexity of splitting large trades across multiple windows to be unfeasible. This gives bots an advantage. This also ensures that prices that are not changed for many windows are given higher weight even if they initially appear to be outliers. Bots also benefit from this new method as they do not have to pay for multiple transactions or recalculate in the event of partial strategy execution.

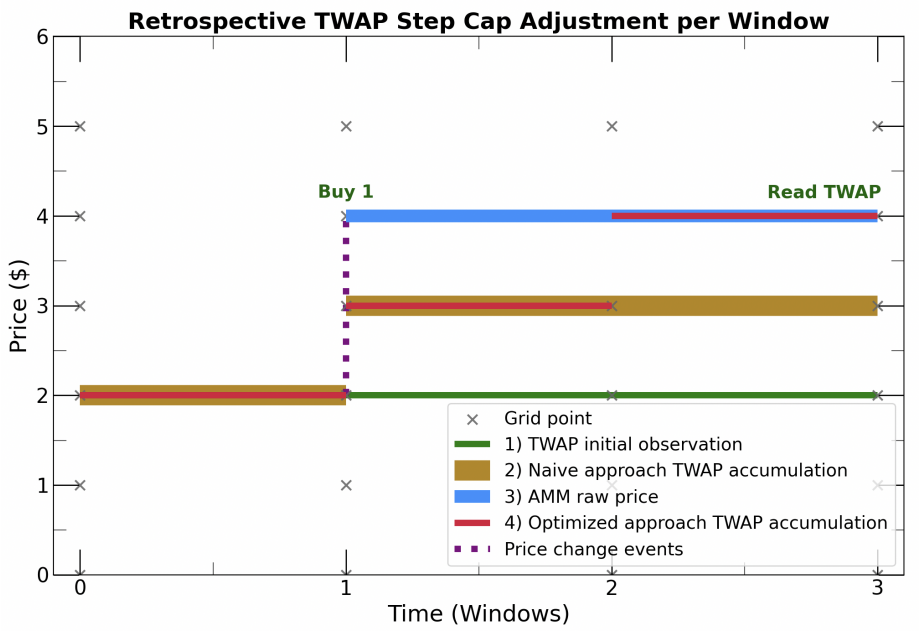

Figure 1: Retrospective TWAP step cap adjustment produces a more accurate TWAP. Line 2 shows the naive TWAP approach, where the step can only increase by the cap once per observation. Line 4 shows a TWAP step cap that can retrospectively increase once per window, regardless of new state updates. This $1 gap in TWAP accumulation would continue indefinitely past window 3, until the next state update. This model uses a TWAP step cap of $1 per window.

5.2 Intra-step TWAP

This method ensures removes race conditions for being the first to land a transaction in a new window. Race conditions lead to MEV opportunities and capital extraction from traders. This new method also allows the TWAP to respond more quickly to changes in price while maintaining the TWAP step cap precision benefits that a large window offers.

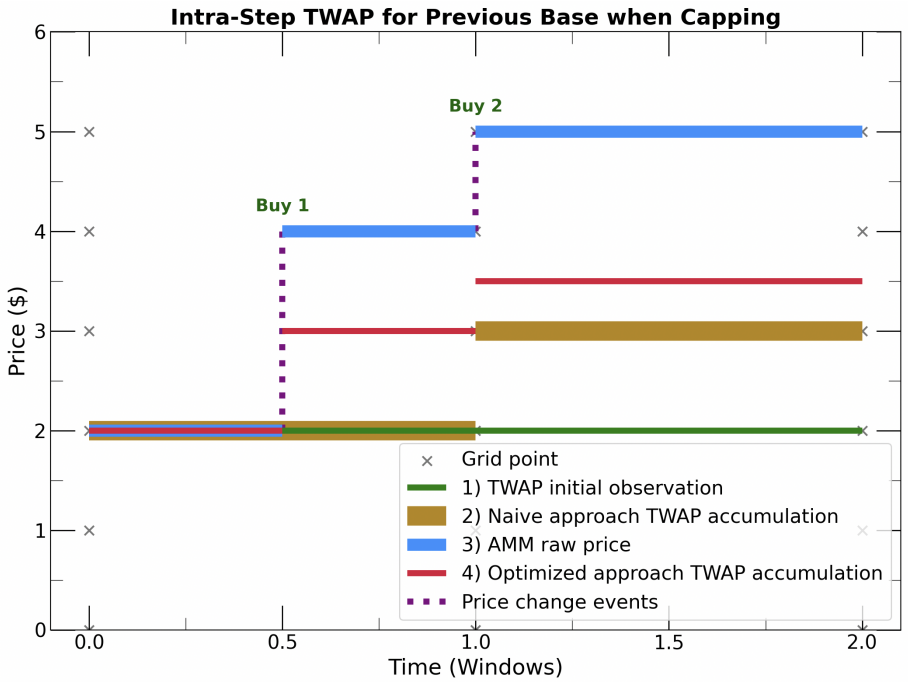

Figure 2: Intra-step TWAP calculation for the previous TWAP base produces a more accurate TWAP. Line 2 shows that the naive TWAP approach has a delay when intra-window price changes occur. It can only update in the next window. Line 4 shows that when a price change occurs, the optimized approach responds immediately while still obeying the TWAP per window max accumulation change cap. The gap in accuracy propagates into the window 1 - 2, where the optimized approach is \$0.5 closer to the raw price. This model uses a TWAP step cap of \$1 per window.

6 Future work

The TWAPs proposed in this paper require an initial TWAP observation and a max TWAP step configs. Future investigation into whether the max TWAP step config can be derived as a fixed percent of the initial TWAP observation may be worthwhile to simplify the total number of configs required. Additionally, for governance markets that runs regularly and have a short duration, deriving the initial TWAP observation from the last market's TWAP may automatically make the TWAP more responsive to rapid price movements.

References

[1] MetaDAOproject. amm.rs – state module for futarchy amm (commit 4092c39). https://github.com/metaDAOproject/futarchy/blob/4092c39/programs/amm/src/state/amm.rs, 2025. Computer program, accessed 2025-05-07.

[2] Curve Finance. Curve technical docs. https://docs.curve.fi/stableswap-exchange/stableswap-ng/pools/oracles/?h=oracle, 2025. Website, accessed 2025-05-07.

[3] Dongbin Bai, Jiannong Cao, Yinfeng Cao, and Long Wen. Ormer: A manipulation-resistant and gas-efficient blockchain pricing oracle for defi, 2024.

[4] MetaDAOproject. Trading statistics – trading interface for metadao. https://metadao.fi/ore/trade-v4/3PCPkFQQo7sU2DxpqnQNEttiyx1t5kByY2WPjCo1qLEZ, 2025. Trading GUI, accessed 2025-05-07.

[5] Mysten Labs. Sui statistics - grafana dashboard for sui validators. https://metrics.sui.io/public-dashboards/4ceb11cc210d4025b122294586961169?from=now-24h&to=now&timezone=America%2FLos_Angeles, 2025. Statistcs dashboard, accessed 2025-05-07.

[6] Solana Labs. Solana explorer – dashboard for solana validators. https://explorer.solana.com/, 2025. Statistcs dashboard, accessed 2025-05-07.