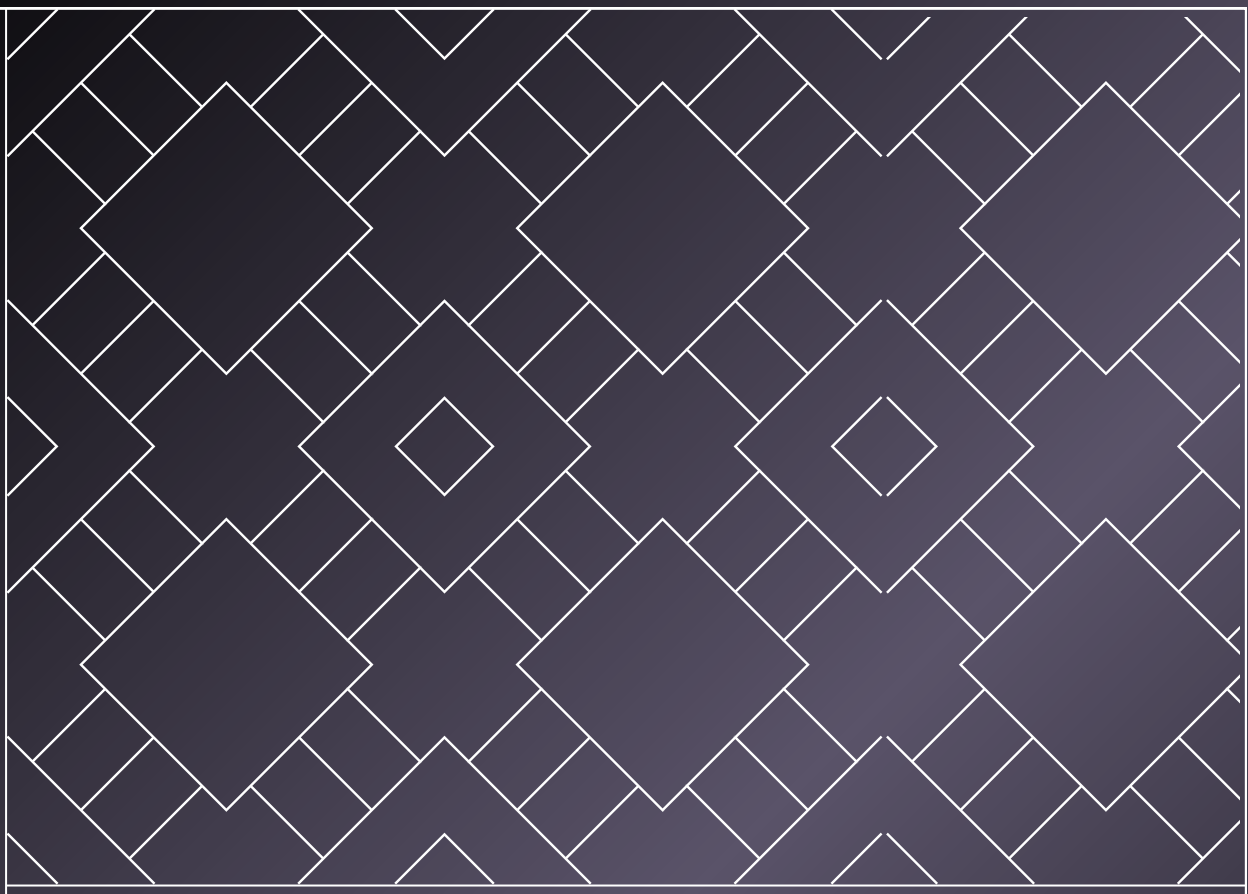


DEVINE OS

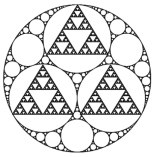
Buffered TWAP with Circuit-Breaker (BTCB)

Quantitative Research And Asset Manager Integrations

Info@Devinegroup.xyz

Buffered TWAP with Circuit-Breaker (BTCB)

Buffered TWAP with Circuit-Breaker (BTCB) in Devine OS



Devine Group

info@Devinegroup.xyz

July 22, 2025

Devine OS Whitepaper Series

Abstract

The Buffered TWAP with Circuit-Breaker (BTCB) method is a framework for managing continuous capital flows in quantitative trading portfolios within the Devine OS blockchain-based vault system. Designed for institutional asset managers, BTCB supports diverse strategies such as trend-following, mean-reversion, and statistical arbitrage. It employs a pre-vault buffer, a 1-hour Time-Weighted Average Price (TWAP) deployment strategy, and a volatility-based circuit-breaker to minimize market impact and safeguard portfolio performance. Built for the Devine OS Arbitrum's HyperliquidVaultRouter contract and an off-chain Manager API, the system tracks buffer capital (Y_t) and deployed capital (X_t), ensuring simplicity, scalability, and compliance with the ERC-4626 standard. Shares are minted and redeemed at live Net Asset Value (NAV), embedding profit and loss (P&L), idle-cash drag, and execution costs to ensure fairness across investors. This paper details the system's architecture, mathematical model, and synchronization mechanisms, offering a clear and comprehensive guide to its operation.

1. Introduction

Managing continuous algorithmic deposit and withdrawal capital flows in quantitative trading portfolios presents unique challenges, particularly in balancing market impact, execution costs, and investor fairness. The Buffered TWAP with Circuit-Breaker (BTCB) method, integrated into the Devine OS ecosystem on Arbitrum, addresses these challenges with a robust and flexible framework. By combining a pre-vault buffer, a 1-hour Time-Weighted Average Price (TWAP) deployment strategy, and a volatility-based circuit-breaker, BTCB minimizes market disruption while supporting diverse trading strategies, such as trend-following, mean-reversion, or statistical arbitrage.

At the core of the system is the HyperliquidVaultRouter, an ERC-4626-compliant contract that interfaces with Hyperliquid L1 to manage deposits, withdrawals, and settlements. The vault delegates trading logic to an exchange wallet (`vaultWallet`) and relies on the Devine OS Manager API to track buffer capital (Y_t) and deployed capital (X_t). This hybrid on-chain/off-chain architecture ensures simplicity, scalability, and compliance with the ERC-4626 standard, which guarantees that old deposits do not affect new ones. This paper outlines the BTCB model's mathematical foundation, system design, and operational efficiency,

providing a clear roadmap for institutional asset managers to integrate with the Devine OS ecosystem.

2. System Architecture

The BTCB framework operates within a cohesive architecture that balances on-chain reliability with off-chain flexibility. Its key components are:

- **Vault Contract**: The HyperliquidVaultRouter, an ERC-4626-compliant contract on Arbitrum, tracks total capital via `latestExchangeBalance`, manages share minting and burning, and processes deposits and withdrawals. It delegates trading to `vaultWallet` and requires admin confirmations for on-chain/off-chain settlements.
- **Manager API**: An off-chain system that tracks Y_t (buffer capital) and X_t (deployed capital), executes TWAP deployments, enforces circuit-breaker logic, and synchronizes with the vault for accurate NAV accounting.
- **Hyperliquid L1**: Executes quantitative trading strategies, with capital transfers coordinated between the exchanges `vaultWallet`, the `HyperliquidVaultRouter` Arbitrum contract and the Devine OS off-chain systems.

2.1. Vault Design

The vault contract (`HyperliquidVaultRouter`) is the system's on-chain backbone, handling:

- **Deposits**: Users deposit USDC, with a 0.2% entry fee sent to `treasuryWallet` and the remainder routed to an exchange `vaultWallet` enabled by Devine OS backend. Deposits are recorded as `PendingDeposit` structs, confirmed via `confirmDeposit`, and shares are minted at the current NAV.
- **Withdrawals**: Users initiate withdrawals via `initiateWithdrawal`, creating a `WithdrawalQuote` with NAV and share data. Withdrawals are processed in batches through `confirmWithdrawalWithPnL`, applying a performance fee on positive P&L.
- **NAV and Liquidity**: The vault tracks total capital as `latestExchangeBalance` ($C_t = X_t + Y_t$), updated algorithmically via `updateVaultExchangeBalance`.

By treating all capital as `latestExchangeBalance`, the vault avoids on-chain tracking of Y_t and X_t, simplifying its logic while delegating complex operations to the Manager API.

2.2. Manager API Role

The Manager API enhances the system's efficiency by handling:

- **State Management**: Tracks Y_t and X_t in for real-time updates.

- **TWAP Deployment**: Executes 1-hour TWAP deployments, transferring capital from Y_t to X_t via `vaultWallet`, tailored to the quantitative trading strategy.
- **Circuit-Breaker**: Pauses deployments when portfolio volatility σ_t exceeds $\sigma_{\max}$, retaining idle capital in Y_t.
- **Synchronization**: Optionally Updates `latestExchangeBalance` to reflect $Y_t + X_t$, ensuring NAV accuracy.

This separation allows the vault to focus on on-chain accounting while the API manages dynamic trading logic, ensuring scalability and adaptability for various quant strategies.

3. Deposit Inflow Process

Deposits flow into the system at a continuous rate λ, modeled as:

$$dD_t = \lambda \, dt. \tag{1}$$

The buffer capital evolves according to:

$$dY_t = dD_t - u_t \, dt, \tag{2}$$

where $u_t \geq 0$ is the deployment rate into the quant portfolio, managed by the Manager API.

In practice, the vault processes deposits as follows:

- Users call `deposit(assets, receiver)`, transferring USDC to the contract. A 0.2% fee is sent to `treasuryWallet`, and the remainder moves to `vaultWallet`.
- Deposits are logged in `userPendingDeposits` with a nonce and timestamp, pending confirmation.
- The Devine OS backend sends to the Devine OS Manager API to update Y_t and coordinates algorithmically to call `confirmDeposit`, minting shares at the current NAV.

This process ensures fairness by minting shares at the precise `assetsPerShare` (NAV), incorporating micro-effects like idle-cash drag and execution costs, as required by ERC-4626.

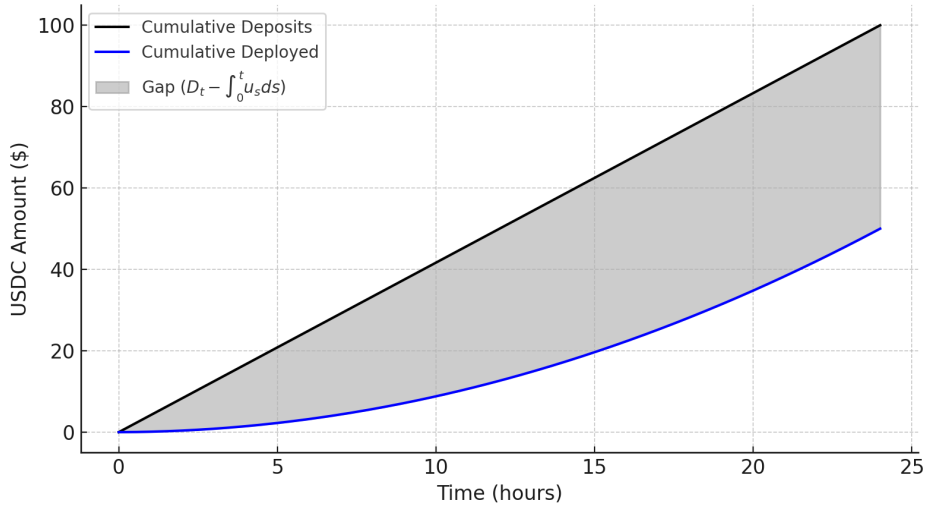


Figure 1: Gap Between Inflows and TWAP Capital Deployment Outflows (0–24 h). The black line shows cumulative USDC deposits at a constant rate over 24 h. The blue line depicts the TWAP-deployed capital. The grey shading represents the buffer Y_t, the gap between inflows of idle capital and active capital outflows.

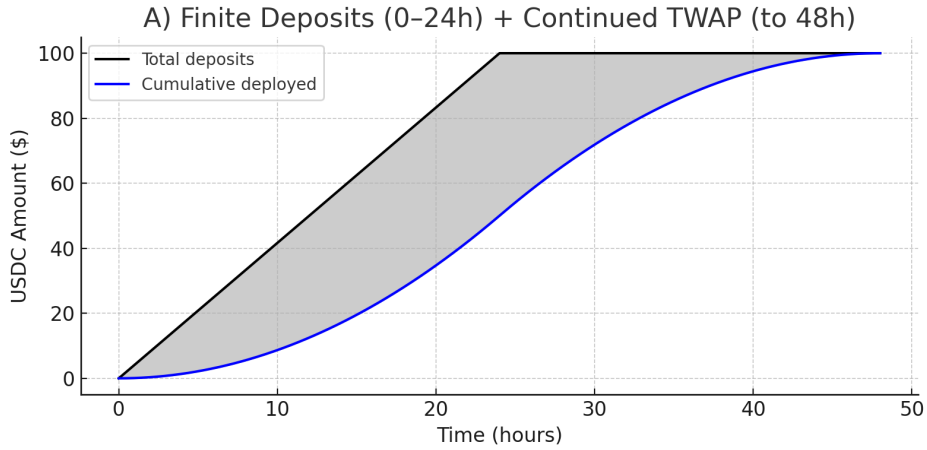


Figure 2: Finite Deposit Window with TWAP (0–48 h). Deposits flow at a constant rate from 0 h to 24 h (black), then cease. TWAP deployment (blue) continues to 48 h, draining the buffer (grey) by hour 48.

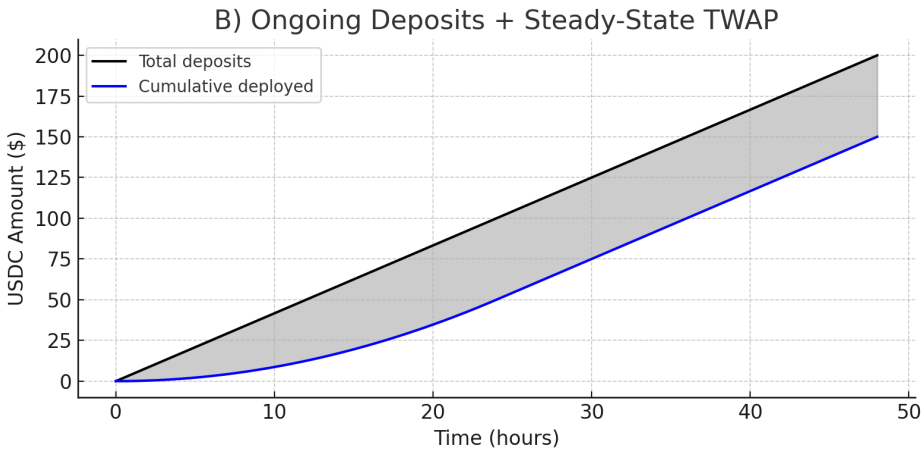


Figure 3: Steady-State Deposits and TWAP (0–48 h). Continuous deposits (black) maintain a constant rate. TWAP outflows (blue) ramp up over 24 h, then match inflows, stabilizing the buffer (grey) as liquidity "in flight."

3.1. Efficiency of the Deposit Process

The deposit process is engineered for efficiency and fairness, with key features:

- **Continuous Flow**: Modeling deposits as a continuous rate λ prevents market disruptions from sudden inflows, mirroring gradual real-world investments.
- **Pre-Vault Buffer**: The buffer Y_t absorbs deposits, smoothing volatility and preventing system overload, as shown in Figures 1–3.
- **TWAP Deployment**: The 1-hour TWAP deploys capital gradually, minimizing market impact. Figures 1 and 2 illustrate how the buffer is drained efficiently.
- **Idle-Cash Minimization**: Rapid deployment reduces idle-cash drag, ensuring capital is utilized effectively without compromising market stability.
- **Scalability**: The process adapts to varying inflow rates and market conditions, with the Manager API adjusting deployments dynamically.
- **Fairness**: Shares minted at NAV ensure equitable treatment, embedding costs and drag proportionally, per ERC-4626 standards.

This design achieves a balance of liquidity, efficiency, and fairness, making it ideal for continuous capital management.

4. TWAP Deployment

The BTCB method employs a 1-hour TWAP to deploy capital from Y_t to X_t, minimizing market impact. For a deposit of size A at time t_0, the deployment rate is:

$$u_t = \frac{A}{\Delta}, \quad t \in [t_0, t_0 + \Delta], \tag{3}$$

where $\Delta = 3600$ seconds (1 hour). For overlapping deposits $\{(t_i, A_i)\}$, the total rate is:

$$u_t = \sum_{i: t_i \le t < t_i + \Delta} \frac{A_i}{\Delta}. \tag{4}$$

The Manager API executes TWAP as follows:

- Confirmed deposits increment Y_t.
- Capital is deployed in small batches over 1 hour via `vaultWallet`, aligned with the quant strategy.
- X_t increases and Y_t decreases accordingly.
- The API or Devine OS backend continuously updates `latestExchangeBalance` to reflect $Y_t + X_t$.

Execution costs are modeled as $c(u_t) = \eta u_t^2$, balanced against idle-cash drag (e.g., 0.0001% per hour at 1% p.a. stablecoin yield). This gradual deployment reduces price impact, making it suitable for diverse trading strategies.

5. Circuit-Breaker Mechanism

To protect the portfolio from volatile markets, the Manager API implements a circuit-breaker that pauses TWAP deployments when short-term volatility exceeds a predefined threshold.

5.1. Volatility Calculation

Portfolio volatility is computed over a rolling window τ:

$$\sigma_t = \sqrt{\frac{1}{\tau} \int_{t-\tau}^{t} \left(r_s - \bar{r}\right)^2 ds}, \quad \bar{r} = \frac{1}{\tau} \int_{t-\tau}^{t} r_s \, ds, \tag{5}$$

where r_s is the instantaneous return on X_s. In practice:

- Returns r_s are derived from high-frequency price ticks from Hyperliquid L1.
- A ring buffer updates $\bar{r}$ and σ_t in real time.

5.2. Circuit-Breaker Logic

The deployment indicator is:

$$I_t = \begin{cases} 1, & \sigma_t \leq \sigma_{\max}, \\ 0, & \sigma_t > \sigma_{\max}, \end{cases} \tag{6}$$

where $\sigma_{\max}$ is a strategy-specific volatility threshold. When $\sigma_t > \sigma_{\max}$:

- $I_t = 0$, pausing new TWAP orders.
- Ongoing TWAP slices confirmed on-chain, but no new capital is deployed.
- Deposits accumulate in Y_t until volatility subsides.

5.3. Adjusted Deployment Rate

The deployment rate becomes:

$$u_t = I_t \min\left(\lambda, \frac{Y_t}{\Delta}\right). \tag{7}$$

If $I_t = 0$, $u_t = 0$, halting deployments.

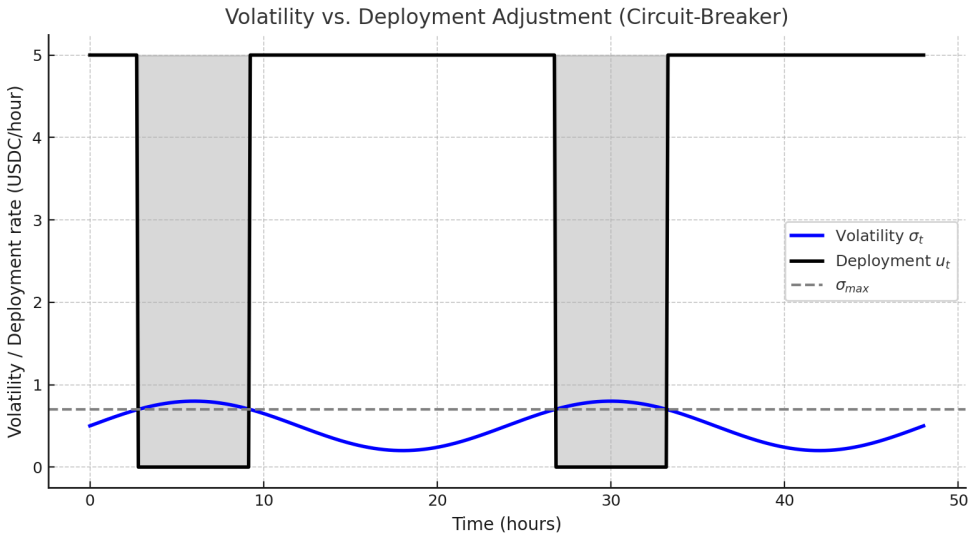


Figure 4: Volatility and Deployment Dynamics. The blue curve shows realized volatility σ_t, the black bars shows deployments u_t (USDC/hour), and the grey dashed line marks $\sigma_{\max}$. Shaded regions indicate pauses when $\sigma_t > \sigma_{\max}$.

5.4. Implementation

The Manager API:

- Streams returns from Hyperliquid L1 for real-time volatility updates.

- Computes σ_t every minute using efficient algorithms.
- Pauses TWAP when $\sigma_t > \sigma_{\max}$, setting $I_t = 0$.
- Logs activations and alerts portfolio managers.

This mechanism preserves capital during turbulent markets, reducing slippage and protecting NAV, as illustrated in Figure 4.

6. Withdrawal Processing

Withdrawals are processed in discrete batches based on the underlying quantitative strategy, updating deployed capital:

$$X_{t_k} \leftarrow X_{t_k^-} - \sum_{w \in \mathcal{W}_{t_k}} A_w, \tag{8}$$

$$Y_{t_k} \text{ unchanged.} \tag{9}$$

The vault handles withdrawals as follows:

- Users call `initiateWithdrawal(shares)`, creating a `WithdrawalQuote` with NAV and share data.
- Devine OS communicates with manager via the API to confirm batch liquidity is available via `confirmBatchUsdcReceived`, updating `batchUsdcRemaining`.
- `confirmWithdrawalWithPnL` finalizes withdrawals, applying a 30% fee on positive P&L.

The Manager API:

- Tracks withdrawal requests, reducing X_t.
- Ensures `batchUsdcRemaining` reflects available USDC.

The FIFO batch process ensures withdrawals at quoted NAV confirmed via the manager.

7. Profit and Loss Dynamics

The portfolio's performance is driven by:

$$dP_t = r_t X_t \, dt - \eta u_t^2 \, dt, \tag{10}$$

where r_t is the return rate on X_t, and ηu_t^2 represents quadratic slippage costs. Total P&L over $[0, T]$ is:

$$P_T = \int_0^T \left(r_t X_t - \eta u_t^2 \right) dt. \tag{11}$$

7.1. P&L Visualization

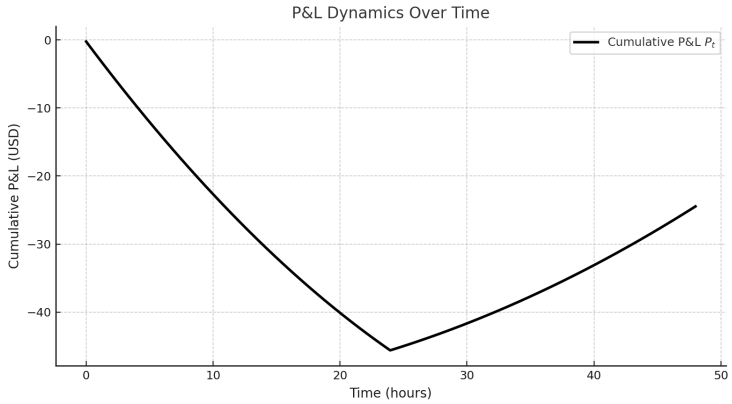


Figure 5: Cumulative P&L P_t. The black curve shows net gains from X_t minus execution costs, rising during active deployment and stabilizing when deployment slows.

7.2. Idle-Cash Drag

Idle capital in Y_t potenitally earning a low stablecoin yield depending on exchange venue, creating drag:

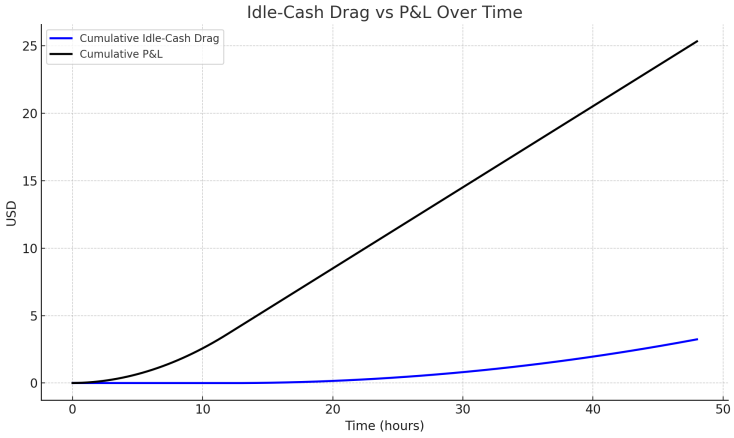


Figure 6: Idle-Cash Drag vs. P&L. The blue curve shows the opportunity cost of Y_t, while the black curve shows P&L from X_t. The shaded gap highlights lost performance from uninvested capital.

7.3. Key Insights

- High u_t drives P&L growth but increases slippage costs (ηu_t^2).
- Paused deployments flatten P_t, with idle-cash drag accumulating.
- The TWAP window Δ balances P&L acceleration against costs.

8. Optimization Objective

The BTCB method maximizes expected P&L:

$$\max_{u_t} \mathbb{E}\left[\int_0^T (r_t X_t - \eta u_t^2)\, dt\right], \tag{12}$$

subject to:

- Circuit-breaker: $u_t = 0$ if $\sigma_t > \sigma_{\max}$.
- Flow constraints: $0 \leq u_t \leq \lambda$.

This objective balances returns, execution costs, and risk protection, adaptable to any quant strategy.

9. Manager API Implementation

The Manager API ensures efficiency and simplicity through:

- **State Management**: Stores Y_t and X_t updating with deposits and deployments.
- **TWAP Deployment**: Queues deposits in Y_t, deploying capital at $u_t = A/\Delta$ over 1 hour via `vaultWallet`.
- **Circuit-Breaker**: Monitors σ_t from Hyperliquid L1 data, pausing TWAP if $\sigma_t > \sigma_{\max}$.
- **Withdrawal Coordination**: Tracks requests, reduces X_t, and ensures liquidity via `confirmBatchUsdcReceived`.

The protocol ensures consistency by updating Y_t, X_t, and `latestExchangeBalance` after deposits, TWAP batches, and withdrawals, with hourly reconciliations.

10. ERC-4626 Compliance

The BTCB method upholds the ERC-4626 invariant, ensuring old deposits do not affect new ones:

- **NAV Calculation**: `totalAssets` reflects `latestExchangeBalance` ($X_t + Y_t$), with shares minted at:

$$\text{Shares} = \frac{\text{Deposit Amount} \cdot \text{Total Shares}_t}{X_t + Y_t}. \tag{13}$$

- **Idle-Cash Drag**: Y_t is included in NAV, shared pro-rata across investors.
- **Execution Costs**: Costs ηu_t^2 are reflected in X_t and NAV.
- **Circuit-Breaker**: Paused deployments increase Y_t, fairly captured in NAV.

- **Withdrawals**: Processed at quoted NAV, isolating new deposits from P&L.

Risks like desynchronization, algorithmic reconciliations or delayed updates are mitigated through regular `latestExchangeBalance` updates, Redis audit logs, and validation against Hyperliquid L1 data within the Devine OS backend.

11. Design Evaluation

Managing Y_t and X_t in the Manager API offers:

- **Simplicity**: Keeps the vault focused on accounting, avoiding complex on-chain logic.
- **Scalability**: Off-chain processing supports high-frequency deployments.
- **Flexibility**: Allows dynamic adjustments to TWAP and volatility parameters.
- **Fairness**: Ensures accurate NAV via synchronization, upholding ERC-4626.

An on-chain alternative increases transparency but raises gas costs and complexity, making the API approach optimal for quant trading strategies.

12. Conclusion

The BTCB method offers a sophisticated yet accessible solution for managing continuous capital flows in quantitative trading portfolios. By integrating a pre-vault buffer, 1-hour TWAP deployment, and a volatility-based circuit-breaker, it minimizes market impact and protects performance across strategies like trend-following and statistical arbitrage. The HyperliquidVaultRouter and Manager API ensure simplicity, scalability, and ERC-4626 compliance, with robust synchronization maintaining NAV fairness. This framework balances returns, costs, and risk, delivering institutional-grade reliability within the Devine OS ecosystem.