Facemelt: A Novel AMM for Unified Spot and Leverage Trading

calmxbt
https://github.com/calmdentist/facemelt

June 2025

Abstract

This whitepaper introduces *Facemelt*, a novel automated market maker (AMM) that allows both spot and leverage trading within a unified liquidity framework for long-tail assets. By employing the Uniswap V2 style invariant with modifications to accommodate leveraged positions, Facemelt enables a rich trading experience while ensuring the underlying AMM remains solvent at all times. Facemelt also allows for asset creation without seed capital for assets that meet certain criteria. We describe the mathematical foundation - operational mechanics for opening and closing leveraged positions and liquidations.

1 Introduction

Existing perpetual DEXs are based on the vAMM model, which resembles traditional perpetual futures contracts built on an AMM. However, this model has several limitations:

- Fragmented liquidity: separate liquidity pool for leverage trading.
- Counterparty risk: Longs require shorts as counterparties to trade, and vice-versa.
- Oracles: Oracles are used to fetch spot prices from external sources. This assumes trust and creates attack vectors.

Facemelt unifies spot and leverage trading within a single liquidity pool - providing deeper liquidity, reducing counterparty risk, and eliminating the need for oracles.

2 Mathematical Foundation

At its core, Facemelt uses a constant product invariant. However, to support a unified market for spot and leverage trading, it introduces the critical concepts of **real reserves** and **effective reserves**.

- **Real Reserves (x_r, y_r)**: The actual amount of SOL and tokens held in the protocol's vaults. These are only ever changed by spot traders depositing or withdrawing assets.
- **Effective Reserves (x_e, y_e)**: The virtual reserves that all trades are priced against. For spot trades, x_e and y_e are changed in lockstep with x_r and y_r. For leverage trades, only the effective reserves are modified, reflecting the virtual nature of the positions.

Leverage is made possible by allowing the effective reserves to deviate from the real reserves. The difference represents the total outstanding debt owed to the pool by leverage traders. This debt is denominated in an abstract, price-agnostic unit of liquidity called **delta k (Δk)**.

2.1 Core Variables

The protocol tracks the following state variables:

- x_r, y_r: The real reserves of the base asset and token.
- x_e, y_e: The live, effective reserves of the base asset and token.
- $\sum \Delta k_{\text{longs}}$: The sum of original Δk from all open long positions.
- $\sum \Delta k_{\text{shorts}}$: The sum of original Δk from all open short positions.
- $I_{\text{long}}(t), I_{\text{short}}(t)$: Side-specific cumulative funding indices used to calculate the state of individual positions.

The fundamental relationship for any swap is against the effective reserves:

$$x_e \cdot y_e = k_e \tag{1}$$

3 Leverage Trading Mechanism

Similarly to traditional derivative contracts, leveraged positions on Facemelt are virtual claims on pool value rather than physical token ownership.

3.1 Opening a Leveraged Position

A leveraged position is created by swapping a notional amount (collateral $\times$ leverage) against the AMM's effective liquidity. The collateral is always denominated in the input asset of the swap, and the position's virtual size is denominated in the output asset.

3.1.1 Long Position (Base Asset as Collateral)

To open a long position on the token, a trader provides collateral c_x in the base asset. The notional value ($c_x \times L$) is swapped virtually to determine the position size, Δy:

$$\Delta y = \frac{y_{\text{pre}} \cdot (c_x \cdot L)}{x_{\text{pre}} + (c_x \cdot L)} \tag{2}$$

This Δy becomes the position's virtual size, denominated in the token asset.

3.1.2 Short Position (Token as Collateral)

To open a short position on the token, a trader provides collateral c_y in the token asset - note that shorting with sol as collateral is equivalent to an atomic spot buy + leverage sell. The notional value ($c_y \times L$) is swapped virtually to determine the position size, Δx:

$$\Delta x = \frac{x_{\text{pre}} \cdot (c_y \cdot L)}{y_{\text{pre}} + (c_y \cdot L)} \tag{3}$$

This Δx becomes the position's virtual size, denominated in the base asset.

In both cases, the output amount is not physically transferred; it is an accounting entry representing a claim on pool value. The borrowed liquidity, Δk, is calculated as the reduction in the constant product caused by the virtual swap. Let x_{post} and y_{post} be the virtual reserves after the notional swap:

$$\Delta k = k_{\text{pre}} - k_{\text{post}} = (x_{\text{pre}} \cdot y_{\text{pre}}) - (x_{\text{post}} \cdot y_{\text{post}}) \tag{4}$$

This Δk value is stored with the position and represents the liquidity borrowed from the pool. The trader must repay this borrowed liquidity through:

1. **Continuous funding payments**: Accrued over the position's lifetime at a rate that increases quadratically with the pool's total leverage
2. **Final settlement**: Any remaining debt is settled when closing the position

4 Side-Specific LP Funding Rate

To compensate liquidity providers for the borrowed liquidity, Facemelt implements **side-specific funding rates** that continuously repay each side's outstanding Δk debt, realizing it as an increase in the pool's effective reserves. Unlike traditional perps that use a single global funding rate, Facemelt isolates the funding curves for longs and shorts. This design encourages contrarian trading: opening a position on the opposite side does not penalize existing positions, and may even reduce their funding costs.

4.1 Funding Rate Formula

Each side's funding rate is a function of the portion of the real reserve that side is currently borrowing. Longs borrow tokens from the pool and shorts borrow SOL, so we define:

$$b_{\text{long}} = y_r - y_e \tag{5}$$

$$b_{\text{short}} = x_r - x_e \tag{6}$$

$$\lambda_{\text{long}} = \frac{b_{\text{long}}}{y_r} \tag{7}$$

$$\lambda_{\text{short}} = \frac{b_{\text{short}}}{x_r} \tag{8}$$

The funding rate per unit time for each side is then:

$$\text{FundingRate}_{\text{long}} = C \cdot \lambda_{\text{long}}^2 \tag{9}$$

$$\text{FundingRate}_{\text{short}} = C \cdot \lambda_{\text{short}}^2 \tag{10}$$

where C is a configurable constant.

4.2 Contrarian Incentive

By denominating utilization in the borrowed asset's real reserve, contrarian trades naturally improve the funding environment for the opposite side:

- When a trader opens a **long**, they deposit SOL collateral, increasing x_r. This directly lowers $\lambda_{\text{short}} = b_{\text{short}}/x_r$, easing funding costs for existing shorts.
- When a trader opens a **short**, they deposit token collateral, increasing y_r. This directly lowers $\lambda_{\text{long}} = b_{\text{long}}/y_r$, easing funding costs for existing longs.

This mechanism rewards liquidity provision from both directions and encourages balanced markets without requiring explicit incentive programs.

4.3 Implementation via Side-Specific Distribution

The protocol continuously and atomically updates its global state based on each side's funding rate. For any time interval Δt, the fees paid by each side are calculated independently:

$$\text{Fees}_{\text{longs}} = \text{FundingRate}_{\text{long}} \cdot (\sum \Delta k^{\text{orig}}_{\text{longs}}) \cdot \Delta t \tag{11}$$

$$\text{Fees}_{\text{shorts}} = \text{FundingRate}_{\text{short}} \cdot (\sum \Delta k^{\text{orig}}_{\text{shorts}}) \cdot \Delta t \tag{12}$$

Note that fees are calculated based on **original** Δk to ensure linear amortization, while the funding rate uses **effective** Δk to reflect outstanding debt.

These fees, which are denominated in units of k, must be converted back into their respective reserve assets to be realized. Long positions borrow the token, so their fee payments increase the effective token reserve. Short positions borrow the base asset, so their payments increase the effective base asset reserve.

$$\Delta y_e = \frac{\text{Fees}_{\text{longs}}}{x_e} \tag{13}$$

$$\Delta x_e = \frac{\text{Fees}_{\text{shorts}}}{y_e} \tag{14}$$

The effective reserves and debt pools are then updated. At the same time, Facemelt maintains two normalized funding indices, $I_{\text{long}}(t)$ and $I_{\text{short}}(t)$, which measure the fraction of the respective side's original Δk that has already been repaid through funding. The incremental updates are:

$$\Delta I_{\text{long}} = \begin{cases} \frac{\text{Fees}_{\text{longs}}}{\sum \Delta k^{\text{orig}}_{\text{longs}}}, & \text{if } \sum \Delta k^{\text{orig}}_{\text{longs}} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

$$\Delta I_{\text{short}} = \begin{cases} \frac{\text{Fees}_{\text{shorts}}}{\sum \Delta k^{\text{orig}}_{\text{shorts}}}, & \text{if } \sum \Delta k^{\text{orig}}_{\text{shorts}} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

Because each increment is expressed as a fraction of original debt, the indices are bounded and never overshoot the remaining principal.

4.4 Global Solvency Floor

To prevent "Phantom Liquidity" (where effective reserves erroneously exceed real reserves due to amortizing underwater positions at depressed prices), Facemelt enforces a strict Global Solvency Floor.

When funding fees are converted from Δk back into effective reserves, the protocol enforces the invariant:

$$R_{effective} \leq R_{real} \tag{17}$$

Specifically:

$$y^{new}_{effective} = \min(y_{effective} + \Delta y_{fees}, y_{real}) \tag{18}$$

$$x^{new}_{effective} = \min(x_{effective} + \Delta x_{fees}, x_{real}) \tag{19}$$

This ensures that the protocol never claims to have more effective liquidity than it physically holds in its vaults. If the calculated fee repayment would exceed this cap (e.g., during a price crash for long positions), the excess repayment is simply discarded. This results in a permanent reduction of the pool's constant product k (burning liquidity) rather than creating insolvent claims. This "fail-safe" mechanism prioritizes the safety of the real reserves over the preservation of the virtual curve's constant product.

4.5 Position Amortization via Funding

Funding payments act as a continuous amortization of each position's virtual debt (Δk) and its corresponding virtual claim on the pool's assets (y_{position}). This process is denominated in units of k, the AMM's constant product, so it remains price-agnostic. As a position accrues funding fees, both its effective size and its effective debt shrink proportionally.

Let $s \in \{\text{long}, \text{short}\}$ denote the market side of a position. The **Position Remaining Factor**, $f_s(t)$, for a position opened at time t_{open} is:

$$f_s(t) = \max\left(0, 1 - (I_s(t) - I_s(t_{\text{open}}))\right) \tag{20}$$

This factor represents the fraction of the original position that remains after accounting for funding payments on side s. It allows us to calculate the effective values at any time t:

- **Effective Position Size**: $y_{\text{effective}}(t) = y_{\text{original}} \cdot f_s(t)$
- **Effective Borrowed Liquidity**: $\Delta k_{\text{effective}}(t) = \Delta k_{\text{original}} \cdot f_s(t)$

As $I_s(t)$ grows, $f_s(t)$ decreases, ensuring that both the position's claim and its debt are paid down over time while never becoming negative.

4.6 Debt Burn Dynamics

Normalizing the funding indices means the remaining debt evolves linearly with the realized funding, rather than following a cubic differential equation. Let D_s^{orig} be the sum of original Δk for side s at the moment a cohort of positions opens, and let $D_s(t)$ denote the effective (outstanding) debt for that cohort. Funding reduces $D_s(t)$ as:

$$\frac{dD_s(t)}{dt} = -D_s^{\text{orig}} \cdot \text{FundingRate}_s(t) \tag{21}$$

and integrating over time yields:

$$D_s(t) = D_s^{\text{orig}} \cdot \max\left(0, 1 - \int_{t_{\text{open}}}^{t} \text{FundingRate}_s(\tau)\, d\tau\right) \tag{22}$$

If the side-specific leverage ratio stays approximately constant at λ_s, the funding rate is also constant ($\text{FundingRate}_s = C\lambda_s^2$), so the decay becomes:

$$D_s(t) = D_s^{\text{orig}} \cdot \max\left(0, 1 - C\lambda_s^2 \cdot (t - t_{\text{open}})\right) \tag{23}$$

In this regime the time required to repay the entire cohort is:

$$t_{\text{zero}} = \frac{1}{C\lambda_s^2} \tag{24}$$

and the time to repay any fraction scales linearly (e.g., 50% of the debt is gone after $0.5/(C\lambda_s^2)$). This preserves the intuition that higher leverage accelerates repayment, while guaranteeing positions decay smoothly and never overshoot their remaining principal.

Importantly, because each side has its own λ_s, the decay rate for longs is independent of short utilization and vice versa. A heavily one-sided market will see that side's debt burn quickly while the minority side enjoys lower funding costs—naturally incentivizing rebalancing flow.

4.7 Dynamic Leverage Ceiling

To prevent a single trader from consuming the entire curve when the pool is already heavily utilized, Facemelt enforces a leverage multiplier ceiling that adapts to the live leverage ratio. Let the leverage ratio be:

$$\lambda = \frac{\sum \Delta k_{\text{effective}}}{k_e} = \frac{\sum \Delta k_{\text{effective}}}{x_e \cdot y_e} \tag{25}$$

and define the configurable parameters:

- $L_{\text{flat}} = 10\times$: maximum leverage multiplier when the pool is flat ($\lambda \approx 0$)
- $L_{\text{floor}} = 1.5\times$: minimum leverage multiplier when the pool is highly utilized
- $\lambda^\star = 0.6$: leverage ratio at which leverage is clamped to L_{floor}
- $p = 2$: convexity that controls how fast leverage decays as utilization grows

The ceiling is computed as:

$$L_{\text{max}}(\lambda) = L_{\text{floor}} + (L_{\text{flat}} - L_{\text{floor}}) \cdot \left(1 - \text{min}\left(1, \frac{\lambda}{\lambda^\star}\right)\right)^p \tag{26}$$

New positions must satisfy $L \leq L_{\text{max}}(\lambda)$ at the time of execution. Because λ grows with outstanding virtual debt, the permissible leverage decays smoothly when the pool is stressed and quickly recovers once positions amortize or close. This complements the quadratic funding rate: funding makes running leverage expensive, while the ceiling ensures the pool cannot be overwhelmed before funding has a chance to act.

5 Closing a Leveraged Position

When a trader closes their position at time t_{close}, the protocol uses the effective position size and effective borrowed liquidity to calculate the final payout. This process ensures that all accrued funding fees are settled implicitly within the closing swap.

The steps are as follows:

1. **Calculate the Position Remaining Factor**: The protocol first computes the remaining factor for the position based on the side-specific funding index:

$$f(t_{\text{close}}) = 1 - (I_s(t_{\text{close}}) - I_s(t_{\text{open}})) \tag{27}$$

2. **Determine Effective Values**: Using this factor, the effective position size ($S_{\text{effective}}$) and effective borrowed liquidity ($\Delta k_{\text{effective}}$) are calculated from their original values.

3. **Calculate Payout**: The final payout is determined by swapping the effective position size against the pool's current effective reserves while accounting for the repayment of the final effective borrowed liquidity.

   For a long position (collateral in base asset, effective size y_e), the payout P_{base} is:

$$P_{\text{base}} = \frac{x_e \cdot y_e - \Delta k_{\text{effective}}}{y_e + y_{\text{effective}}} \tag{28}$$

   For a short position (collateral in token asset, effective size x_e), the payout is also in the base asset (SOL). The protocol calculates the amount of SOL required to restore the constant product invariant (x_{repay}) and returns the remainder to the user:

$$x_{\text{repay}} = \frac{\Delta k_{\text{effective}}}{y_e} \tag{29}$$

$$P_{\text{base}} = x_{\text{effective}} - x_{\text{repay}} \tag{30}$$

Upon closing, the position's original Δk is removed from its respective debt pool, and the effective reserves are updated. For shorts, only the x_{repay} amount is added back to the effective base reserve.

6 Liquidation Mechanism

A position becomes liquidatable when its payout is less than or equal to 5% of its gross value. However, to prevent manipulation-based liquidations, Facemelt employs an **EMA Price Oracle** that blocks liquidations when rapid price movements are detected.

6.1 EMA Oracle Protection

The protocol tracks an exponential moving average (EMA) of the price with a 5-minute half-life. Before allowing liquidation, it checks:

$$\frac{P_{\text{EMA}} - P_{\text{spot}}}{P_{\text{EMA}}} \leq 10\% \tag{31}$$

If the spot price has diverged more than 10% below the EMA, liquidation is blocked. This prevents atomically profitable manipulation attacks where an adversary crashes the price temporarily to liquidate positions, then immediately reverts the price.

6.2 Liquidation Conditions

A position can be liquidated if and only if:

1. The payout is positive (position not underwater)
2. The payout $P \leq 5\%$ of the position's gross value
3. The EMA divergence check passes (no manipulation detected)

The liquidator's reward is the entire remaining payout. Upon liquidation, the pool's state is updated using the same logic as closing a position, with the effective Δk removed from the global debt tracking.

7 Limbo State

A position enters "limbo" when it meets the liquidation condition but no liquidator has closed it. It can exit limbo if price movements cause recovery: $x_e \cdot y_{\text{effective}} > \Delta k_{\text{effective}}$. Underwater positions still pay the LP funding rate, and thus their effective debt and size will amortize to 0 over time.

8 Liquidity Provisioning and Launching

Facemelt enables permissionless token launches with zero seed capital, using a bonding curve model similar to pump.fun. Traders who buy the token on the bonding curve essentially fund the LP once the token reaches the bonding threshold and graduates to the AMM.

8.1 Fair-Launch Bonding Curve

To ensure a fair and transparent launch, Facemelt employs a fixed-supply, linear bonding curve. The process is as follows:

1. **Total Supply Deposit**: The token creator mints the entire fixed supply (e.g., 420 million tokens) and deposits it into a program-controlled vault. This transparently commits all tokens to the launch mechanism, with no creator pre-allocation.
2. **Linear Bonding Curve**: A predetermined number of tokens (e.g., 280 million) are designated for sale on a linear bonding curve, where the price is a function of the number of tokens sold: $P = m \cdot Q_{\text{sold}}$. The curve is engineered to raise a fixed amount of a base asset (e.g., 200 SOL). The creator participates on the same terms as any other buyer.
3. **AMM Graduation**: The final purchase that meets the SOL target automatically triggers the AMM launch. The protocol uses 100% of the raised SOL to seed one side of the liquidity pool. The other side is seeded with the remaining tokens from the initial deposit (e.g., 140 million), which is mathematically guaranteed to be exactly half the number of tokens sold.

This mechanism deterministically results in a healthy, liquid market from the moment of creation, typically achieving a 1:1 ratio of total liquidity to circulating market cap. For example, a launch raising 200 SOL will create a liquidity pool valued at 400 SOL, against a circulating market cap of 400 SOL.

9 System Properties

Facemelt's design provides several important properties:

9.1 Solvency

The protocol is always solvent because the pricing mechanism ensures that the output can never exceed the real reserve, regardless of input size.

9.2 No Seed Capital Requirement

The bonding curve based token launch mechanism allows early traders to become liquidity providers, and eliminates the need for external LPs.

9.3 Economic Incentive Alignment

The quadratic funding rate and effective reserve model create powerful incentive alignment:

- **Leverage traders**: Continuous debt repayment through funding incentivizes closing positions, especially when the funding rate is high due to heavy pool utilization.
- **Spot traders & LPs**: Benefit from the continuous growth of the effective reserves, which are fed by the funding payments from leverage traders. This leads to deeper liquidity and better execution prices over time.
- **Protocol**: The dynamic funding rate creates a natural economic cap on total system leverage. As total Δk grows relative to the effective liquidity k_e, the funding rate accelerates, making new leverage prohibitively expensive.

10 Considerations and Future Work

Facemelt makes the tradeoff of temporarily reducing effective liquidity for traders in order to power leverage. This liquidity is restored through the LP funding rate and settled when positions are closed or liquidated. This tradeoff was chosen because in order to not reduce effective liquidity for traders, solvency would not be guaranteed in extreme cases (bank runs).

However, if liquidations were integrated at the protocol level (i.e. swaps liquidate positions that would become underwater), effective liquidity could remain constant. This is an ideal scenario - designing such a liquidation mechanism that can stay within the limits of compute usage in smart contracts is very difficult, if possible.

Future work will involve designing a liquidation engine integrated at the protocol level, and exploring other fundamentally different AMM mechanisms (concentrated liquidity/liquidity bins for example) that create a net improvement to the protocol.