TURBIN3 BUIDLERS COHORT — CAPSTONE 2026

# V-AMM

## A Volatility-Adaptive Automated Market Maker on Solana

An autonomous AMM that adjusts its curve shape and fee schedule based on realized on-chain volatility — no oracles, no governance, no keeper keys.

**Sribatsha Dash** · vamm.srivtx.xyz

`75yCYNeZrSoVKWk5kFti7tRpRacZHptmAqtPwfc9U4Zt`

Anchor 1.0.1 · Solana Devnet · Rust · June 2026



# Contents

1. Abstract

2. Introduction

3. Background: AMMs and Their Limitations

4. The V-AMM Design

5. Architecture

6. The Three Mechanisms

7. On-Chain Implementation

8. Security Analysis

9. Frontend and Simulation

10. Deployment Status

11. Conclusion

References

# 1. Abstract


### Abstract

We present V-AMM (Volatility-Adaptive Automated Market Maker), a Solana program that autonomously adjusts its bonding curve and fee schedule in response to realized on-chain volatility. Unlike existing AMMs that use fixed amplification parameters and static fee tiers set at launch, V-AMM continuously measures volatility from its own trade history using an Exponentially Weighted Moving Average (EWMA) engine, then maps that signal to both the StableSwap amplification A and a dynamic swap fee ranging from 5 bps to 100 bps.

The system implements a complete 2-asset StableSwap AMM with six instructions. All volatility computation uses integer arithmetic (u128) with a 1e12 fixed-point scale — no floating-point operations, no external oracles, no admin keys. The A parameter ramps gradually over 9000 slots (~1 hour) to prevent curve-transition arbitrage, while fees are EMA-smoothed and rate-limited to 10 bps per slot.

The program is deployed on Solana Devnet, executes swaps within ~100,000 compute units, and is accompanied by a browser-based simulator at vamm.srivtx.xyz running the same mathematical model in TypeScript.


AMM StableSwap Solana DeFi On-chain Volatility EWMA Dynamic Fees Anchor

# 2. Introduction

## 2.1 Motivation

Automated Market Makers (AMMs) have become the dominant decentralized exchange mechanism since the introduction of Uniswap's constant product formula in 2018. The design has evolved through concentrated liquidity (Uniswap v3), StableSwap curves (Curve), and custom bonding functions. Yet one fundamental limitation persists across nearly all deployments: **pool parameters are configured at launch and rarely adapt**.

A StableSwap pool launched with amplification A = 2000 operates with the same flat curve during a market crash as it does during calm periods. The pool cannot sense that volatility has risen and steepen its curve to protect liquidity providers. Similarly, a fee tier chosen at pool creation cannot respond to changing market conditions without a governance vote or privileged admin intervention.

## 2.2 Research Question

> **Research Question**
>
> Can an AMM autonomously adjust its curve shape and fee schedule using only on-chain data — without external oracles, governance processes, or admin keys — while remaining computationally feasible within Solana's constraints and resistant to manipulation?

## 2.3 Contributions

1. An **on-chain EWMA volatility engine** using dual ring buffers (15-minute and 1-hour windows), implemented entirely in u128 integer arithmetic with a 1e12 fixed-point scale.
2. A **volatility-to-curve mapping** that translates realized volatility into a target StableSwap amplification A, with gradual ramping over 9000 slots (~1 hour).
3. A **volatility-to-fee mapping** using piecewise smoothstep interpolation ensuring C1 continuity, followed by EMA smoothing and a 10 bps per-slot rate limit.
4. A **complete Solana program** with six instructions, ~100K compute units per swap, permissionless cranks, and full PDA-based account architecture.
5. An **open-source browser simulator** executing the same mathematical model in TypeScript.

# 3. Background: AMMs and Their Limitations

## 3.1 From Constant Product to StableSwap

The constant product formula (x * y = k), introduced by Uniswap, guarantees that the pool never fully drains. Curve Finance's StableSwap addresses this by blending constant-sum and constant-product behaviors through an amplification parameter *A*:

$$4A(x + y) + D = 4AD + D^3 / (4xy)$$

## 3.2 The Static Parameter Problem

**Table 1:** Static Parameter Mismatch

| Market State | What LPs Need | What a Static Pool Does |
|---|---|---|
| Calm (σ ≤ 15%) | High A, low fee | Same A, same fee — suboptimal |
| Volatile (σ ≥ 75%) | Low A, high fee | Same flat curve, low fee — LP loses |

## 3.3 Governance Is Too Slow

Existing parameter adjustment mechanisms rely on governance votes (Curve A ramp) or privileged keeper accounts. Governance operates on timescales of days; markets move in seconds. V-AMM eliminates both by making parameter updates permissionless and fully on-chain.

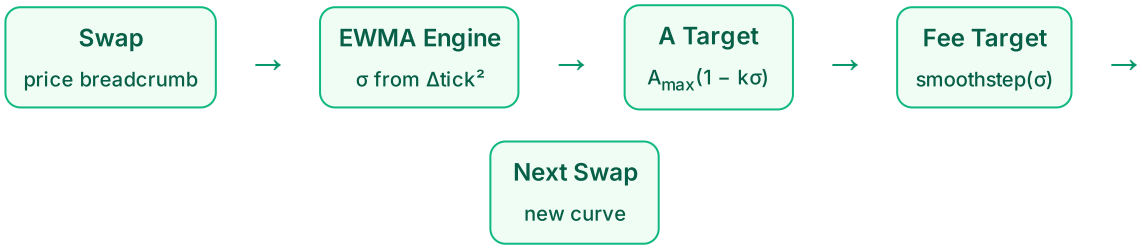
# 4. The V-AMM Design

## 4.1 Design Principles

**Table 2:** V-AMM Design Principles

| Principle | Implementation |
|---|---|
| **Autonomous** | No oracle, no governance, no admin key. Two permissionless cranks. |
| **Gradual** | A ramps over ~1 hour (9000 slots). Fees slide at max 10 bps per slot. |
| **Manipulation-resistant** | EWMA with λ=0.95: single trade moves needle ~5%. |
| **Efficient** | Swap uses ~100K CUs. Newton-Raphson converges in 4-6 iterations. |
| **Composable** | Standard SPL tokens and Anchor PDAs. LP shares are regular SPL tokens. |

## 4.2 The Feedback Loop




## 4.3 Market Regime Response

**Table 3:** Pool Behavior by Market Regime

| Regime | σ Range | A Behavior | Fee |
|---|---|---|---|
| Calm | σ ≤ 15% | High A — flat curve | 5 bps |
| Transition | 15% < σ < 75% | A ramping down | 5 → 30 bps |
| Volatile | 75% ≤ σ < 120% | Low A — steep curve | 30 → 100 bps |
| Extreme | σ ≥ 120% | A near minimum | 100 bps cap |

# 5. Architecture

## 5.1 Program Structure

```
vamm/
├── programs/vamm/src/
│   ├── lib.rs              # Entry point — routes 6 instructions
│   ├── state.rs            # PoolState, VolatilityState, PositionState
│   ├── math/mod.rs         # StableSwap + VolatilityEngine
│   ├── instructions/
│   │   ├── initialize_pool.rs
│   │   ├── swap.rs
│   │   ├── add_liquidity.rs
│   │   ├── remove_liquidity.rs
│   │   ├── update_volatility.rs
│   │   └── update_curve.rs
│   ├── error.rs
│   └── constants.rs
└── Anchor.toml
```

## 5.2 Account Model (PDAs)

**Table 4:** PDA Account Structure

| Account | Seeds | Purpose |
|---|---|---|
| PoolState | `["pool", mint_a, mint_b, pool_id]` | Reserves, curve A, fee state (~416 B) |
| VolatilityState | `["volatility", pool_state]` | EWMA variance, price buckets (~387 B) |
| PoolAuthority | `["authority", pool_state]` | Signer for vaults — no private key |
| LpMint | `["lp_mint", pool_state]` | Fungible LP token mint |
| VaultA / VaultB | `["vault_a"/"vault_b", pool_state]` | Token accounts holding reserves |
| Position | `["position", pool, user, 0]` | LP shares and fee snapshots (~130 B) |

## 5.3 Instruction Set

**Table 5:** Program Instructions

| Instruction | Access | Description |
|---|---|---|
| `initialize_pool` | Public | Create pool PDAs, LP mint, token vaults. |
| `swap` | Public | Execute StableSwap trade; write volatility breadcrumb. |
| `add_liquidity` | Public | Deposit pair; receive LP shares proportional to D. |
| `remove_liquidity` | Public | Burn LP; withdraw proportional reserves + fees. |
| `update_volatility` | Permissionless | Recompute A target and fee from EWMA. |
| `update_curve` | Permissionless | Interpolate A between start and target. |

# 6. The Three Mechanisms

## 6.1 StableSwap Curve with Dynamic A

$$4A(x + y) + D = 4AD + D^3 / (4xy)$$

```
d = sum of reserves
for iteration in 0..64 {
    d_prev = d
    d_p = d * d / (n * n * x * y)
    numerator = ann * sum + d_p * n * d
    denominator = ann * d + (d_p - 1) * (n + 1)
    d = d * numerator / denominator
    if |d - d_prev| ⩽ 1 { break }
}
```

Unlike Curve where A is static, V-AMM's `curve_a_current` interpolates between `curve_a_start` and `curve_a_target` over 9000 slots (~1 hour). When volatility changes by more than 10%, a new ramp begins from the current interpolated value.

## 6.2 EWMA Volatility Engine

$$\text{tick} \approx \log\&sub2;(\text{price}) \cdot 6931 / 10000 \quad \text{(via leading_zeros)}$$

$$r^2 = (\Delta\text{tick})^2 \cdot [\ln(1.0001)]^2$$

$$\text{EWMA}_t = 0.95 \cdot \text{EWMA}_{t-1} + 0.05 \cdot r^2$$

$$\sigma = \sqrt{\text{EWMA}} \cdot \sqrt{(31{,}536{,}000 / 900)} \quad \text{(clamped to 0–500\%)}$$

## 6.3 Dynamic Fee Schedule

**Table 6:** Fee Computation by Volatility

| Condition | Fee (bps) |
| --- | --- |
| $\sigma \le 15\%$ | 5 |
| $15\% < \sigma < 75\%$ | $5 + 25 \cdot S((\sigma - 15\%) / 60\%)$ |
| $75\% \le \sigma < 120\%$ | $30 + 70 \cdot S((\sigma - 75\%) / 45\%)$ |
| $\sigma \ge 120\%$ | 100 |

# 7. On-Chain Implementation

## 7.1 Swap Execution

1. **Validate:** amount_in > 0, pool has reserves, correct mints.
2. **Sync curve:** Interpolate A to current slot.
3. **Calculate fee:** `fee = amount_in * current_fee_bps / 10000`.
4. **Solve StableSwap:** Newton-Raphson for D, then Y (max 64 iterations).
5. **Slippage check:** `output ≥ min_amount_out`.
6. **CPI transfer in:** User → Vault (signed by user).
7. **CPI transfer out:** Vault → User (signed by PoolAuthority PDA).
8. **Update reserves** and accrue fees (90% LP, 10% protocol).
9. **Write volatility breadcrumb:** tick, EWMA update, 15-min bucket.

## 7.2 Volatility Update (Permissionless Crank)

1. Read EWMA variance from VolatilityState.
2. Annualize to σ (clamped 0–500%).
3. Map σ → target_A = $A_{max}$ · (1 − k·σ), min A = 1.
4. Map σ → raw_fee via smoothstep (5–100 bps).
5. EMA-smooth: fee = 0.9 · old + 0.1 · raw.
6. Rate-limit: |fee − current| ≤ 10 bps per slot.
7. If |target_A − current_A| / current_A > 10%: start new ramp (9000 slots).

## 7.3 Liquidity Accounting

LP shares use the D-invariant: first depositor gets shares = D(reserves); subsequent deposits get (ΔD / $D_{old}$) · total_lp_shares. Fees use cumulative fee-per-share accounting — each PositionState stores a `fee_growth_global` snapshot; uncollected fees = (current_global − snapshot) · shares.

# 8. Security Analysis

**Table 7: Adversarial Analysis**

| Attack | Vector | Mitigation |
|---|---|---|
| Volatility manipulation | Wash trades to inflate σ | EWMA λ=0.95: single trade moves ~5%. Volume-weighted stats. |
| Sandwich | Front-run during high-A | Slippage protection + 9000-slot ramp. |
| Curve transition arb | Buy before ramp, sell after | A changes 0.11% per slot. |
| Fee manipulation | Spike σ to 100 bps | 10 bps/slot rate limit + EMA lag. |
| Compute exhaustion | Force max iterations | 64-iteration cap. Early convergence. |
| Oracle griefing | N/A — no external oracle | All data endogenous. |

## 8.2 Defense in Depth

**Layer 1 — Oracle hardening:** Volume-weighted stats and dual time-window cross-verification. **Layer 2 — Transition safety:** Gradual A ramping over 9000 slots. **Layer 3 — Rate limiting:** Fee changes capped at 10 bps/slot. **Layer 4 — Circuit breakers:** Emergency steepening on large single-trade impact.

# 9. Frontend and Simulation

A companion web application at vamm.srivtx.xyz runs the same mathematical model in TypeScript. The simulator uses canonical Curve math; the on-chain Rust has three documented deviations (D_p overflow, compute_y precision, convergence threshold) documented with proposed fixes.

# 10. Deployment Status

```
Program ID: 75yCYNeZrSoVKWk5kFti7tRpRacZHptmAqtPwfc9U4Zt
Network: Solana Devnet | Binary: 447,472 bytes | Slot: 463,918,472
Authority: 6vTYvzm1dwqJreoYHEiUqitnfyDoLmp3seVX6fjFwVFp
```

**Table 8: Development Stack**

| Tool | Version |
|---|---|
| Anchor CLI | 1.0.1 |
| Solana CLI | 3.1.15 |
| Rustc | 1.95.0 |
| TypeScript / Next.js | 5.x / 14.x |

# 11. Conclusion

V-AMM demonstrates that an automated market maker can autonomously adapt its curve shape and fee schedule using only on-chain data. By computing realized volatility through an EWMA engine and mapping that signal to both the StableSwap amplification A and a dynamic fee, the pool responds to market conditions without requiring external oracles, governance votes, or admin keys.

The gradual ramp mechanism (9000 slots for A, 10 bps/slot rate limit for fees) eliminates arbitrage windows while maintaining responsiveness. All operations fit within Solana's compute budget, with swaps executing in approximately 100,000 compute units. The full source code and simulator are available at github.com/srivtx/vamm-capstone.

# References


1. Egorov, M. (2019). *StableSwap: Efficient Mechanism for Stablecoin Liquidity*. Curve Finance Whitepaper.
2. Adams, H., Zinsmeister, N., & Salem, M. (2020). *Uniswap v2 Core*. Uniswap Technical Whitepaper.
3. Adams, H., et al. (2021). *Uniswap v3 Core*. Uniswap Technical Whitepaper.
4. Milionis, J., Moallemi, C. C., Roughgarden, T., & Zhang, A. L. (2022). Automated Market Making and Loss-Versus-Rebalancing. *ACM EC 2022*.
5. Angeris, G., Evans, A., & Chitra, T. (2020). When does the tail wag the dog? Curvature and market making. *arXiv:2012.08040*.
6. Yakovenko, A. (2018). *Solana: A New Architecture for a High Performance Blockchain*. Solana Whitepaper.


---

**V-AMM** · Solana Devnet · Built by **Sribatsha Dash**

vamm.srivtx.xyz · github.com/srivtx/vamm-capstone