# TraceSVM: A Framework for Reverse-Engineering and Analysing Solana's Proprietary AMMs

Florian Klein[0009−0000−5416−3508], Ferdinando Samaria[0009−0001−2045−6190], and Frank Stajano[0000−0001−9186−6798]

Department of Computer Science and Technology, University of Cambridge, UK
fbk24@cantab.ac.uk, fs105@cam.ac.uk, professor+tracesvm@stajano.com

**Abstract.** Proprietary AMMs now account for over half of Solana's DEX volume, yet their pricing logic is compiled into closed-source bytecode and has never been systematically studied. Anyone wishing to analyse these protocols, whether as a researcher, protocol designer, or regulator, faces the same obstacle: the part that matters most is invisible. We present TraceSVM, a reusable, open-source methodology for recovering closed-form quoting formulas directly from deployed sBPF binaries, requiring no operator cooperation and no source code. Our approach combines instrumented local execution, backward dataflow reconstruction and DTA++-style control-dependency resolution. We validate it on synthetic programs of known formula, then apply it to the seven highest-volume Solana propAMMs, recovering validated state-to-quote functions for all seven, with zero observed mismatch. Recovering the formulas allows us to present the first empirical microstructure study of propAMM venues, with concrete findings on quoting styles, inventory regimes, and what drove operator profitability in a 10-hour SOL/USDC session.

**Keywords:** DeFi · Automated market makers · Binary analysis · Solana · Market microstructure · Reverse engineering.

## 1 Introduction

In 2025 Solana settled roughly $1.95 trillion in decentralised exchange (DEX) volume, surpassing Ethereum for much of the year [15]. That volume is routed by aggregators, most notably Jupiter [17], which query every venue and route each swap to the best quote.

A new class of protocol has displaced the open-source, constant-function market makers (CFMMs) that long dominated on-chain trading. *Proprietary AMMs* (propAMMs) are closed-source programs whose pricing logic is compiled to

∗ In Proceedings of CBT 2026, 10th International Workshop on Cryptocurrencies and Blockchain Technology, co-located with ESORICS 2026, Rome, Italy, 17 September 2026. To appear in Springer LNCS. Authors' preprint, accepted by the publisher but before their typesetting.

bytecode and deployed on-chain without source code or documentation. An off-chain operator funds the pool, updates quoting parameters as the market moves and takes the entire trading profit-and-loss, becoming more a traditional market maker than a passive liquidity pool. PropAMMs grew from negligible volume before October 2024 to over 50% of total Solana DEX volume by late 2025, peaking above 65% in early 2026 [15], while on-chain SOL/USDC spreads compressed from roughly 30 basis points to below 1 basis point.

The opacity of propAMM pricing has direct practical consequences for traders who cannot verify quotes against pool state, for operators who cannot diagnose their own spreads, and for researchers who cannot study adverse selection without knowing the pricing function. Although every byte of account state is on the public ledger, the function that maps that state to a quote is not.

There is a gap in the existing literature: propAMMs fall outside existing empirical AMM research (which assumes the pricing function is known) and outside Solana binary analysis research (which typically targets security properties rather than mathematical semantics). The protocols carrying most of Solana's DEX volume remain unexplored. We close this gap from both sides, asking two research questions: (**RQ1**) how can closed-form quoting formulas be recovered from deployed, closed-source sBPF programs and validated against live on-chain behaviour? (**RQ2**) how do propAMMs quote, manage inventory risk, and earn, and what determines their profitability?

*Contributions.*

1. *TraceSVM, a reusable formula-recovery methodology and open-source toolchain* (Section 3). Every input the quoting function reads is a public account field, and the binary runs on every validator. We combine instrumented local execution, backward dataflow tracing from output to account-state inputs, and control-dependency resolution adapting Kang et al.'s DTA++ [18] to a backward setting. The methodology applies to any on-chain program whose output is computed from public account state. TraceSVM is open source at `https://github.com/TraceSVM`.

2. *Recovered closed-form quoting formulas for seven dominant Solana propAMMs* (Section 4). We recover state-to-quote functions for all seven highest-volume protocols, behaviourally validated at 20,000 comparisons per pool per run with zero observed mismatch. The formulas fall into two architectural families (effective-price models and tier-consumption models) and converge, across seven independently operated closed-source programs, on oracle-driven mid-prices, convex size impact, direction asymmetry, and staleness-dependent fees.

3. *A first empirical case study of propAMM behaviour* (Section 5). The recovered formulas convert raw account-state recordings into quotes, spreads, and inventory positions. Replaying them over a 10-hour recording of 288,875 swaps ($107.5M) across all seven protocols, we measure quoting styles, inventory

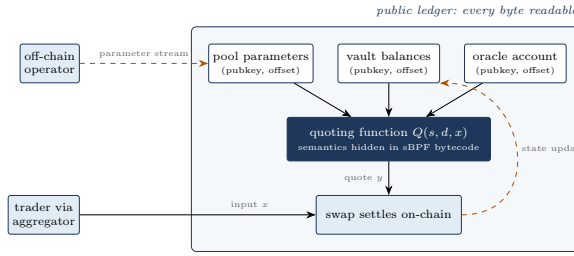



**Fig. 1.** Everything a propAMM reads and writes is a public account field and its bytecode is replicated to every validator; only the map from inputs to quote is hidden. Recovering $Q$ is the one remaining gap.

dynamics, multi-horizon markout, and PnL. Our central finding is that, in this window, profitability tracked order-flow composition rather than volume or formula sophistication; individual operator rankings, however, depend on the hedging benchmark.

## 2 Background and Related Work

To see why propAMMs fall between these two literatures, consider where each design keeps its pricing logic: a CFMM publishes its invariant (e.g. the constant-product rule), whereas a propAMM hides it inside a closed-source program deployed on-chain as a smart contract (Fig. 1).

### 2.1 From Constant-Function to Proprietary AMMs

A CFMM pool holds reserves $x, y$ of two assets and prices trades with a fixed invariant; the constant-product rule $x \cdot y = k$ of Uniswap v2 [4] yields output $\Delta y = y - k/(x + \Delta x)$ and marginal price $y/x$. Arbitrageurs trade the pool to the external market price; the value they extract is a cost to LPs that Milionis et al. [22] formalise as loss-versus-rebalancing (LVR). Concentrated liquidity [5] improves capital efficiency but keeps the invariant fixed and public. These designs are *passive*: the pricing function cannot react to market conditions, so price discovery is delegated to arbitrageurs.

PropAMMs replace the fixed invariant with pricing the operator controls directly. The program still accepts swap instructions, but its quote derives from an operator-set mid-price that tracks external markets, plus operator-tuned spread, depth, and fee parameters stored in on-chain accounts and updated continuously. In our data, operator parameter updates account for 59 to 91% of all pool transactions. Because the operator is the sole capital provider and claims the whole PnL, a propAMM is a latency-aware market maker whose algorithm runs on-chain. Oracle-priced, operator-parameterised AMMs are not new: DODO [10], Lifinity [21], and the broader DEX-survey class [26] prefigure them. Those

designs read an external oracle and re-parameterise infrequently, whereas today's propAMMs crank parameters several times per slot; what is new is recovering and measuring these *closed-source*, high-frequency operators.

Two features of Solana shape the setting. First, execution is near-continuous: 400 ms slots with first-come-first-served inclusion let operators reprice between and within slots. Second, order flow is aggregator-mediated: venues receive a swap only when they quote best at routing time, so spread competitiveness determines flow composition.

### 2.2 sBPF Execution Model

Solana programs are stateless: they read and write state only through accounts passed into each invocation, and all account data is publicly readable. Programs are compiled to sBPF, a register-based eBPF variant with 11 general-purpose registers and explicit call/return conventions [27], and stored on-chain as ELF binaries that anyone can download. At invocation, the runtime serialises the metadata and data of every passed account into a contiguous input region of the VM's address space; a program reads pool state by offset arithmetic from this buffer. Consequently (i) the full quoting logic of a propAMM is contained in public bytecode, and (ii) every input the logic can consult is an account field whose address can be mapped back to a (public key, byte offset) pair. Our methodology exploits both properties.

### 2.3 Related Work

Empirical AMM studies analyse open-source protocols: Lehar and Parlour [20] decompose Uniswap LP returns via markout; Heimbach et al. [14] show concentrated-liquidity provision demands active management at elevated risk; Cartea et al. [7] show Uniswap V3 LPs have on average traded at a loss.

On the program-analysis side, EVM decompilers recover control flow and pseudocode [13] but not closed-form formulas; Solana tools target vulnerabilities on MIR [9] or fuzz compiled binaries [23]. REMaQE [24] recovers equations from native binaries via symbolic execution, but targets single equations and faces path explosion on the branch-heavy logic of real quoting functions. Our backward tracing is instead a dynamic backward slice [25] over a concrete sBPF execution trace, with control dependencies resolved by DTA++-style alternative-path execution [18]. The broader DEX survey [26] classifies propAMMs as a distinct class displacing the open CFMM and CLMM designs that dominate prior survey work. No prior work recovers mathematical semantics from sBPF, and no empirical study of propAMMs exists; this paper addresses both gaps.

## 3 Recovering Quoting Functions from sBPF Bytecode

If the mechanism is hidden in bytecode but every input it reads is a public account field, recovery reduces to tracing how those inputs flow to the quote. We

model each propAMM as a quoting function

$$Q\colon \mathcal{S} \times \{A{\to}B,\ B{\to}A\} \times \mathbb{N} \to \mathbb{N}, \qquad y = Q(s, d, x), \tag{1}$$

mapping observable on-chain state $s \in \mathcal{S}$ (vault balances and pool parameters), the swap direction $d$ between the pool's two tokens $A$ and $B$, and the input amount $x$ to the output amount $y$. TraceSVM recovers the quoting function through a four-stage pipeline: controlled execution (Section 3.1), tracing (Section 3.2), backward reconstruction with control-dependency resolution (Sections 3.3 and 3.4), and validation (Section 3.5).

### 3.1 Local Simulation of the Deployed Binary

Recovering $Q$ requires executing the on-chain program with controlled inputs against real state; on mainnet that would need funded transactions and would mutate the pool. We instead fork all accounts a pool requires (state, token vaults, mints, oracles) from mainnet RPC in a single atomic call, inject them with the program's dumped `.so` binary into a fork of the LiteSVM [2] test VM, and execute swaps locally. For this, three things are important. *Clock warping*: several protocols embed staleness checks that abort when the runtime slot disagrees with slot values stored in account data, so the VM clock is warped to the snapshot slot. *Production-path dispatch*: swaps are routed through the aggregator program via cross-program invocation, matching how production flow reaches each protocol. *State isolation*: account state is restored between swaps, so a sweep over input amounts yields ground-truth $(x, y)$ pairs at a fixed state. Humidifi and SolFi V2 obfuscate state by XOR-ing account fields against pads embedded in the binary's read-only data; the pads are extracted from the ELF and applied during decoding.

### 3.2 Execution Tracing

We instrument the sBPF interpreter's execution loop: each call to `step()` is wrapped to capture pre- and post-execution register state, operand values, and memory accesses. Every instruction execution receives a monotonic sequence number (`exec_seq`), so loop iterations at the same program counter remain distinguishable. For every consumed value the tracer records a `source_def` backward link (the `exec_seq` of the instruction that produced it) by reading the per-register definition table *before* the current instruction's definition is written. A dataflow analyser classifies each produced value's origin and tracks memory stores in an address-keyed table, so loads resolve to the most recent store at that address; `sol_memcpy` syscalls are modelled as synthetic stores to preserve provenance through buffer copies. After execution, the runtime produces a table mapping each VM address range to the originating account, so the analysis resolves a raw load to the account and byte offset it came from.

### 3.3 Backward Formula Reconstruction

Given a trace and the output value $y$, we reconstruct the computation as a dynamic backward slice (Fig. 2, left). Starting from the last instruction that

```
INPUTS (2 of 14): [IN] fp_val = 1568301972, fee_mult = 154  @pool +0x348,+0x354
COMPUTATION (excerpt of 33 steps):
  t21    = (t18 * fee_mult) // 2560000 = 60156       # fee on t18 = 10^9 (input)
  t23    = t18 - (t21 + 1)             = 999939843   # net of fee
  t28    = (849183970 + t23 * fp_val) >> 24          # Q-format / 2^24
  result = t28 // 1000                 = 93472458    # USDC out (6 dp)
VERIFICATION: PASS  (all 33 intermediate values match the trace)
```

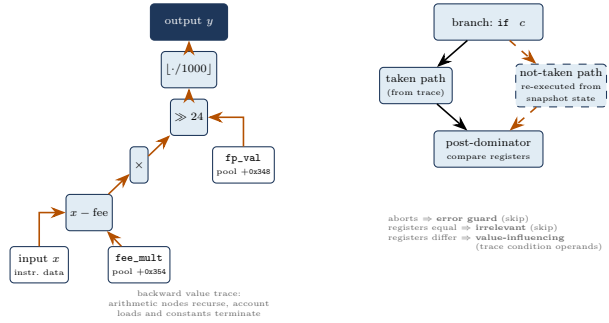



**Fig. 2.** Top: recovered trace of a real 1 SOL BisonFi swap; re-execution reproduces every intermediate value. Bottom left: the same computation as a backward-reconstructed graph. Bottom right: control-dependency resolution; each not-taken path is executed to the branch's post-dominator, then classified by its effect.

wrote $y$, the algorithm recursively follows `source_def` edges. Arithmetic and bitwise operations recurse into both operands and become graph nodes; loads from account data, constants, and instruction-data arguments terminate as tagged leaves; stack traffic, register moves, function returns, and memcpy chains are pass-throughs that the traversal walks across without adding nodes. The result is a topologically sorted pseudocode listing from leaves to $y$ (Fig. 2); re-evaluating it with the same wrapping 64-bit unsigned arithmetic the VM uses reproduces every intermediate value in the trace.

A pseudocode skeleton is not yet a formula: its arithmetic shape must be interpreted and its leaves named. We match recurring assembly idioms (Table 1): `(x*c)»K` reads as Q-format fixed-point multiplication $x \cdot c/2^K$, and shift-and-or limb assembly builds coefficients wider than 64 bits. Leaf semantics come either from known referents (vault balances, the clock sysvar) or from context during iteration. This interpretive step is the one part of the pipeline that remains human-in-the-loop: the toolchain proposes structure, and we hypothesise the formula that explains it. Semantic labels (e.g. "inventory capacity", "staleness penalty") are inferred from context; they are partially validated structurally by the spread-vs-size correspondence in Fig. 3(a), but are not independently verified by the soak loop.

### 3.4 Control-Dependency Resolution

Dataflow alone misses *control* dependencies: for example, a tier-selection branch determines which fee multiplier is used, but only the multiplier's value appears

in the dataflow. We resolve these with two cooperating analyses (Fig. 2, right). Statically, we parse the ELF, build the control-flow graph, and compute each branch's immediate post-dominator with the Cooper, Harvey, and Kennedy algorithm [8] on the reversed CFG. Dynamically, we adapt DTA++ [18] to a backward setting: for each branch near a traced computation, a minimal sBPF interpreter executes the *not-taken* path from the trace's captured register state until the post-dominator, an abort, or a 500-instruction limit. Branches that hit the limit are treated as value-influencing, the conservative choice; syscalls on the alternative path are approximated as no-ops, which can misclassify branches with side-effecting alternatives, though this is rare given sBPF's register-heavy calling convention. Branches whose alternative path aborts are error guards; those reaching the post-dominator with identical registers are irrelevant; both are pruned. Branches whose alternative path produces differing registers are value-influencing: they enter the recovered formula as case distinctions, and their condition operands are themselves traced backward, with control analysis temporarily disabled to prevent recursive explosion.

### 3.5 Behavioural Validation

Each recovered formula is implemented as a standalone arbitrary-precision quoter and validated in two ways. *Soak testing* runs 100 iterations against the live chain; every iteration fetches a fresh mainnet snapshot (new vault balances, operator parameters, and oracle values) and compares the quoter against the real binary on 100 input amounts spanning 1 to 100% of vault balances in both directions: 20,000 comparisons per pool per run at $\pm 1$ token-unit tolerance. No mismatch was observed over the tested state distribution. Each mismatch is saved as a self-contained regression snapshot; diagnosis binary-searches the input axis for the smallest failing amount, re-traces it, and diffs the pseudocode against the quoter's intermediate values, so each failure becomes a patch plus a permanent regression test. Binary upgrades are detected by hash monitoring and trigger re-validation. *Synthetic ground truth.* As an integration test of the toolchain, we ran it unmodified against three sBPF programs we compiled with known formulas (constant-product, fixed-rate-with-fee, two-tier piecewise); in every case the recovered pseudocode matched the source, confirming the tracing is correct where we know the answer.

**Table 1.** Recurring assembly idioms across the seven recovered protocols.

| Trace shape | Interpretation |
|---|---|
| `(x*c)»K` | Q-format fixed-point multiply $x \cdot c/2^K$ |
| `(hi«64)\|lo` | multi-limb assembly of a wide (e.g. 128-bit) coefficient |
| `(x+d-1)//d` | ceiling division $\lceil x/d \rceil$ |
| `x*r//d; x-...` | fractional fee deduction |
| `min(rem, V*m/10^4)` | inventory-bounded tier consumption |
| `(clk-slot)*rate` | staleness delta scaled into a fee penalty |

*Limitations.* Our validation shows that each quoter agrees with its binary on every state we tested; it does not prove that the formula is complete for every input. The reason is that recovery follows concrete execution paths, so a branch we never triggered is a branch we never saw. Without source code to compare against, the soak loop is the closest thing we have to a coverage measure: it resamples live state on every iteration and sweeps trade size from 1 to 100% in both directions, which spans the region these pools quote in, but not beyond it. The claim also stops at the state-to-quote function. Three protocols layer transaction-context checks on top of it (Section 5.3); those read inputs outside the domain of $Q$ and are not part of what we validated. A branch- or block-coverage figure over the swap handler would not remedy this: it would mostly measure the surrounding account-handling and dispatch scaffolding, in a ratio not comparable across the seven binaries; the soak test's resampling of live state and full-range size sweep is the coverage argument we can make.

*Availability.* We release the instrumented interpreter, the tracing and reconstruction toolchain, the recovered quoters, and the synthetic test programs as open source at `https://github.com/TraceSVM`. A tagged snapshot is archived at `https://doi.org/10.5281/zenodo.22047225`; it includes the capture pipeline, so an equivalent account-update recording can be reproduced from a fresh mainnet stream. The snapshot names each analysed binary by its on-chain program ID. Applying the method to a further protocol needs no change to the tracer or the reconstruction; the work that remains is naming the recovered leaves and running the soak test.

## 4 Recovered Quoting Mechanisms

We chose the seven protocols by Solana DEX volume share in early 2026; all are closed-source, with SOL/USDC pools and downloadable binaries. We recovered and validated all seven state-to-quote functions, and they share a common structure; we present each family below through one representative protocol. The full specification of every protocol is its executable quoter in the TraceSVM repository (`propamm-simulator/scripts/<protocol>/quoter.py`), next to the analysed binaries, which are named by program ID, and a README that walks through the recovery workflow.

### 4.1 Two Architectural Families

Table 2 summarises the recovered designs. In all seven, the mid-price $P$ is set by the operator through oracle writes or parameter updates, not derived from an invariant, which reduces the structural LVR cost of passive AMMs [22]; residual adverse selection may still arise from oracle latency and stale parameters. The families differ in how the output is assembled.

**Table 2.** Architectural features of the seven recovered propAMMs. Group A applies a single adjusted price to the full input; Group B consumes it across discrete depth tiers.

| Protocol | Group | Output structure | Staleness | Inventory | Asymmetric |
|---|---|---|---|---|---|
| AlphaQ | A | $y = x \cdot P_{\text{eff}}$ | | ✓ | ✓ |
| Humidifi | A | $y = \lfloor x \cdot P / D_{\text{eff}} \rfloor$ | | ✓ | ✓ |
| SolFi V2 | A | $y = y_{\text{raw}} \cdot (1 - \phi)$ | ✓ | ✓ | ✓ |
| BisonFi | B | $\sum_i y_i$ (pivot tiers) | ✓ | | ✓ |
| Tessera V | B | $\sum_i y_i$ (position tiers) | | | ✓ |
| GoonFi V2 | B | $\int \pi(p)\, dp$ (position tiers) | ✓ | ✓ | ✓ |
| ZeroFi | B | $\sum_i y_i$ (observation tiers) | ✓ | | ✓ |

*Group A: effective-price models.* AlphaQ, Humidifi, and SolFi V2 apply one adjusted price to the whole input. AlphaQ is representative: a direction-specific midpoint multiplier $Q_d$ is adjusted for trade size and inventory,

$$Q_{\text{adj}} = Q_d \mp \alpha\sqrt{x} \mp \beta x^2 + \text{sgn}(r)\left(\gamma\sqrt{|r|} + \delta r^2\right), \tag{2}$$

where $r$ is the pool's remaining inventory capacity and Greek letters are operator-set fixed-point parameters; the $\mp$ terms widen the spread convexly with size, the inventory terms discount trades that restore balance, and a flat safety price derived from a separate operator-set multiplier caps the output. Humidifi composes a base spread, two size-impact terms, and a rebalancing rebate (applied only when the trade reduces imbalance) into a fixed-point accumulator that scales an effective denominator $D_{\text{eff}}$; SolFi V2 corrects the oracle price for vault imbalance and applies a fee that widens with both oracle age and trade size.

*Group B: tier-consumption models.* BisonFi, Tessera V, GoonFi V2, and ZeroFi consume the input across discrete depth tiers at progressively worse prices, $y = \sum_i y_i$, giving the operator direct control over the depth schedule. BisonFi is representative: after a fee $\phi$ including a staleness penalty $\rho \cdot \max(0,\, s_{\text{clock}} - s_{\text{pool}})$, the net input is consumed across tiers priced at signed offsets from the mid,

$$y = \sum_i \left\lfloor \frac{\min(x_{\text{rem}},\, w_{i,\text{base}}) \cdot P_i}{2^{48} \cdot S} \right\rfloor, \qquad P_i = \left\lfloor \frac{P\,(10^4 + \text{adj}_i\,\alpha)}{10^4} \right\rfloor, \tag{3}$$

with $\text{adj}_i \in \{-4, \ldots, +4\}$ and tier widths set as operator-chosen fractions of the quote reserves, converted to base units as $w_{i,\text{base}} = \lfloor w_i \cdot S \cdot 2^{48} / P_i \rfloor$. Tessera V tracks a cumulative position that depletes tier widths across directions; GoonFi V2, the most complex recovered design, branches among three quoting paths on two slot-age counters, decaying its stored position when state is fresh and falling back to an input-weighted stale quote otherwise; ZeroFi computes per-tier fees from two operator-updated observations using bit-exact software floating point, with no on-chain inventory logic at all.

### 4.2 Convergent Design Features

The seven protocols converge on the same core mechanisms, counted here over the soak-verified numerical behaviour rather than the semantic labels of Section 3.3, which are our interpretations. All seven use an oracle-driven mid-price, convex trade-size impact (via $\sqrt{x}$, $x^2$, or piecewise tiers), and separate buy/sell parameterisation; a majority also widen fees with staleness and adjust pricing for inventory in the formula itself. That seven independently operated, closed-source binaries share this market-making structure (size impact, inventory skew, signal-staleness pricing [6,12]) has three possible explanations we cannot distinguish: independent rediscovery of standard market-making designs, a shared ancestry, or mutual imitation (operators reverse-engineering each other, as we did). All three are consistent with the selection pressure of aggregator routing, which rewards whichever venue quotes best. The designs differ in where strategy lives: SolFi V2 embeds inventory response in the formula, while ZeroFi leaves it to the operator's off-chain process, a robustness-versus-flexibility trade-off we return to in Section 6.

## 5 Empirical Case Study

The recovered formulas and account layouts convert raw account bytes into quotes, spreads, and inventory positions. We use this to ask what separates the operators: the pricing mechanism, or the off-chain strategy we cannot observe.

*Dataset.* We streamed account updates for 17 SOL/USDC pools of the seven protocols through a non-voting validator [1] with a gRPC geyser plugin [3], capturing intra-slot timestamps and the causing transaction signatures (distinguishing swaps from deposits and withdrawals). A continuous 10-hour session yielded 2.2M state updates, 288,875 detected swaps, and $107.5M volume across all seven protocols; SOL/USD traded in a tight $66.4 to $67.5 range, i.e. low volatility. This is a case study, not a representative sample. The window covers one pair, one volatility regime, and one part of the trading day, so what follows describes how these venues behaved during it rather than how propAMMs behave in general. Tessera V shares vaults across its pools; we recover its per-pool flow from on-chain position counters, so it joins the swap-level analysis. Swaps were detected per transaction as opposite-sign changes in a pool's two vault balances, gated to the pool's own accounts to avoid contamination from shared vaults and multi-hop aggregator legs. Table 3 summarises activity and quoting per protocol.

### 5.1 Quoting Behaviour and Market Segmentation

Replaying each recovered quoter at every recorded state update yields the spread each protocol offered at any size and instant. Quoting styles differ: BisonFi and Humidifi quote the tightest resting spreads, near 0.4 and 2.2 bps; Humidifi and GoonFi V2 spike in swap slots (to ∼27 and ∼34 bps) as their staleness-gated

**Table 3.** Activity and quoting over the 10-hour window, sorted by median *quoted* spread at $100. †AlphaQ's thin SOL/USDC pool is kept in the dataset but excluded from spread and markout comparisons.

| Protocol | Spread (bps) | Swaps | Swap share | Volume | Vol. share | TVL |
|---|---|---|---|---|---|---|
| BisonFi | 1.9 | 176,375 | 61.1% | $72.6M | 67.6% | $4.94M |
| Humidifi | 2.4 | 38,174 | 13.2% | $10.7M | 10.0% | $1.11M |
| SolFi V2 | 4.6 | 21,421 | 7.4% | $8.9M | 8.3% | $3.19M |
| Tessera V | 5.5 | 24,496 | 8.5% | $11.4M | 10.6% | $2.39M |
| ZeroFi | 5.5 | 3,212 | 1.1% | $0.5M | 0.5% | $0.97M |
| GoonFi V2 | 5.7 | 23,547 | 8.2% | $3.2M | 3.0% | $362K |
| AlphaQ† | 101.9 | 1,650 | 0.6% | $0.1M | 0.1% | $52K |
| **Total** | | **288,875** | | **$107.5M** | | **$13.0M** |

fee floors and position tiers activate; Tessera V and ZeroFi hold near-constant ∼5.5 bps bands; SolFi V2 sits at ∼4.6 and widens under directional flow. Slippage is convex for every protocol, and its shape exposes the recovered family: smooth curves for effective-price models, discrete steps where tier consumption binds (Fig. 3(a)). Spread competitiveness maps onto flow (Fig. 3(b)): BisonFi's tight spread and $4.9M book win a plurality at every size, from ∼29% of sub-$10 swaps to ∼78% above $10K and 67.6% of volume, while Humidifi and GoonFi V2 compete only on small trades and Tessera V on the largest.




(a) quoted spread vs. trade size

(b) flow share by trade size

**Fig. 3.** (a) Median *quoted* two-sided spread vs. size (log-log, AlphaQ excluded): smooth convex curves for effective-price models, discrete steps where tier consumption binds. (b) Share of swaps in each size bin per protocol.

## 5.2 Inventory Dynamics

Dealer theory predicts active inventory management [6], and the recordings show distinct regimes. Some operators hard-wire inventory control into the formula: SolFi V2 holds the tightest imbalance band ($q(t) = (v^{\$}_{\text{base}} - v^{\$}_{\text{quote}})/\text{TVL}$, $\sigma = 1.7\%$) through a vault-imbalance correction that also widens its spread under directional flow. Others leave it off-chain: ZeroFi's formula carries no inventory

term and quotes a near-constant spread, managing its book through the operator's parameter stream alone. Several venues run a persistent net-short base position rather than reverting to zero, consistent with net buy-side flow over the window that the operator carries as inventory instead of continuously hedging out. Mean-reversion half-lives fitted to the imbalance series span more than an order of magnitude, from a couple of minutes for the tightest, highest-frequency venues to over an hour for the widest: faster reversion co-occurs with tighter spreads and higher swap frequency, because a tighter spread leaves a thinner margin to absorb an adverse price move, so these venues cannot hold an open position for long and rebalance quickly.

### 5.3 Order-Flow Composition and Adverse Selection

Markout is the standard measure of execution quality: the move in the reference mid-price over the seconds to minutes after a fill. For each swap $i$ at execution price $P_i^{\text{exec}}$ we compute the operator's markout [20] at horizon $\tau$,

$$\text{markout}_i(\tau) = d_i \cdot \frac{M_{t_i+\tau} - P_i^{\text{exec}}}{M_{t_i}} \times 10^4 \quad [\text{bps}], \tag{4}$$

where $d_i = +1$ for a base-to-quote swap, in which the operator *buys* base, and $d_i = -1$ for a quote-to-base swap, in which it *sells*. With this convention a positive markout means the operator bought below, or sold above, the price the market subsequently settled on, earning the spread net of drift; negative markout is adverse selection, the counterparty better-informed. $M_t$ is the *consensus mid*: the median mid-price across the active propAMM pools at time $t$, computed from the replayed quoters wherever at least three contribute. We use it here and in the realised-spread decomposition below; the PnL of Section 5.4 instead marks to the Pyth SOL/USD oracle as an independent reference. Because all seven protocols ultimately track the same external price, this reference shares a common signal with the venues it evaluates. Two features limit the residual endogeneity. First, each protocol's markout uses a *leave-one-out* consensus that excludes that protocol's own pools from the median, so no venue is benchmarked against itself. Second, re-marking against the Pyth SOL/USD oracle, fully independent of the consensus, preserves the pattern. Nine horizons from 1 s to 600 s avoid dependence on a single threshold.

Equal-weighted, every active protocol earns positive per-trade markout at 5 s (Fig. 4), led by ZeroFi and Humidifi at around 1.1 bps and lowest for the volume leader BisonFi. Volume-weighting compresses every protocol toward zero and turns SolFi V2 and Tessera V, the mid-spread venues that carry large flow, slightly negative; the signs are reference-sensitive, but the pattern is consistent with the continuous-hedging PnL of Section 5.4 for every venue except Tessera V, whose ranking is the most benchmark-sensitive. The same gradient holds within each protocol: markout falls from the smallest to the largest trade-size quintile, several venues turning negative on their largest trades. Large trades thus erode

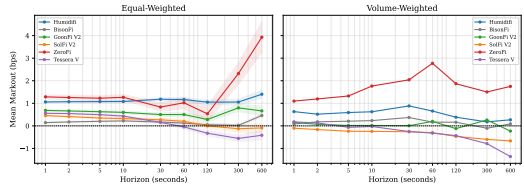



**Fig. 4.** Operator markout by horizon (positive = operator profit); left equal-weighted, right volume-weighted; shaded bands are 95% circular block-bootstrap CIs. AlphaQ excluded.

the operator's edge while small trades are profitable to quote against, the long-standing prediction of the asymmetric-information literature [19,11], which a realised-spread decomposition [16] confirms. Some operators defend off the formula as well: replayed at each fill, our quoter reproduces BisonFi's and Humidifi's executions almost exactly but only 19 to 34% of SolFi V2, GoonFi V2, and ZeroFi fills, which clear *better* than the bare formula. This is not evidence of a wrong formula but of an extra input outside the domain of $Q$ in Eq. (1): these venues read transaction-level signals (the Instructions sysvar, a `jitodontfront` routing sentinel, competitor-program checks) to quote tighter for aggregator-routed, likely-retail flow and worse for flow that looks like arbitrage, such as a Jupiter route whose input and output mints coincide.

### 5.4 Operator Profitability

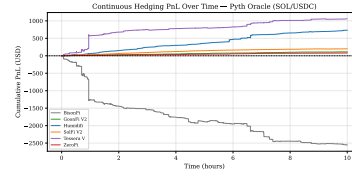



**Fig. 5.** Cumulative continuous-hedging operator PnL by protocol, marked to the Pyth oracle.

We aggregate flow composition into PnL in the spirit of the LVR continuous-hedging framework [22], marking every inventory change to the Pyth SOL/USD oracle across the nine active pools,

$$\Pi_{\mathrm{CH}} = \sum_i \Big[ \Delta v_{q,i} + \Delta v_{b,i} \cdot P_{\mathrm{ref}}(t_i) \Big] \tag{5}$$

where $\Delta v_{q,i}$ and $\Delta v_{b,i}$ are the quote-token and base-token vault balance changes at transition $i$, $P_{\mathrm{ref}}$ is the Pyth oracle price, and decimal scaling to USD is omitted. It is an upper bound on hedged profit, ignoring hedging frictions (exchange fees,

transfer latency). A static-hedge endpoint metric [14] serves as a robustness check.

Under continuous hedging, Tessera V and Humidifi earned the most (+$1,058, +$732; Fig. 5); GoonFi V2, SolFi V2, and ZeroFi sat modestly positive; BisonFi, the volume leader, was the only clear loss (−$2,552), stepped down by individual large trades. The static endpoint metric flips the two heavy books (BisonFi to +$1,948, Tessera V to −$827), so their ranking is sensitive to how inventory drift is marked. All three metrics agree qualitatively: the tightest-spread operator without large flow (Humidifi) is consistently profitable, the venue with the most large flow (BisonFi) gives much of its edge back, and no operator earns an outsized return here. In this window, the composition of order flow mattered more than its quantity.

## 6 Discussion

*Mechanism vs. strategy.* PropAMMs sit between passive AMMs and market makers: the on-chain formula is the quoting *mechanism*, the operator's parameter stream the *strategy*. Designs split this intelligence differently (hard-wired in the formula for SolFi V2, delegated to the operator for ZeroFi, or replaced by spreads tight enough to attract self-balancing flow for Humidifi), yet all seven converge on the same classic market-making model (Section 4.2), plausibly under shared aggregator selection pressure with limited mutual visibility. A self-reinforcing loop may sustain this: tight spreads win routing, the routed retail flow is profitable to quote against, and that profit sustains the tight spreads.

*Limitations.* The empirical window is a single 10-hour, low-volatility session on one pair (SOL/USDC); results may differ in trending markets, on other pairs, or outside Solana's market structure. Our analysis abstracts from Solana's private ordering (priority-fee auctions, Jito-style bundles, and alternative block builders such as BAM), which favours co-located operators and shapes timing our cross-section does not capture. The recovered formulas are snapshots of upgradeable programs (hashes monitored; no upgrades during recording) and need not hold for future versions. Backward tracing operates on concrete executions, not all inputs, and its register-only post-dominator comparison could in principle miss memory-mediated branch effects.

*Ethics and disclosure.* This study used only public data: account state broadcast by the Solana network and program binaries that anyone can download from the ledger. We sent no transactions to any production pool; every experiment ran against forked state in a local VM, so no operator's capital, quoting, or availability was affected. We sought pricing semantics, not vulnerabilities, and we report no exploitable defect. Publishing the recovered formulas removes an asymmetry that currently favours whoever has already reverse-engineered a competitor: traders and researchers gain a verification ability that operators already have. Most propAMM operators are anonymous deployments with no public contact channel; we notified those we could reach before publication.

## 7 Conclusion

PropAMM pricing, though hidden in bytecode, can be recovered from the public inputs each program reads and the binary every validator runs, and checked against the live program. We answered **RQ1** by delivering TraceSVM, a reusable methodology and open-source toolchain that reconstructs closed-form quoting formulas from deployed sBPF binaries with no operator cooperation required, validated here against seven dominant Solana protocols with zero observed mismatch. We answered **RQ2** by replaying those recovered formulas over 288,875 live swaps in a single SOL/USDC session: in that window profitability tracked order-flow composition rather than volume or formula sophistication, and adverse selection concentrated in large trades, though the ranking of individual operators depends on how inventory drift is marked. The same pipeline applies to any on-chain program that computes its output from public account state. We invite other researchers to apply it, extend it, and build on the recovered formulas. TraceSVM is available at https://github.com/TraceSVM.

## References


1. Anza-xyz/agave. anza-xyz (Apr 2026), https://github.com/anza-xyz/agave
2. LiteSVM/litesvm. LiteSVM (Apr 2026), https://github.com/LiteSVM/litesvm
3. Rpcpool/yellowstone-grpc. Triton One RPC (Apr 2026), https://github.com/rpcpool/yellowstone-grpc
4. Adams, H., Zinsmeister, N., Robinson, D.: Uniswap v2 Core. Whitepaper, Uniswap (Mar 2020), https://app.uniswap.org/whitepaper.pdf
5. Adams, H., Zinsmeister, N., Salem, M., Keefer, R., Robinson, D.: Uniswap v3 Core. Tech. rep., Uniswap Labs (Mar 2021), https://app.uniswap.org/whitepaper-v3.pdf
6. Avellaneda, M., Stoikov, S.: High-frequency trading in a limit order book. Quantitative Finance **8**(3), 217–224 (Apr 2008). https://doi.org/10.1080/14697680701381228
7. Cartea, Á., Drissi, F., Monga, M.: Decentralized Finance and Automated Market Making: Predictable Loss and Optimal Liquidity Provision. SIAM Journal on Financial Mathematics **15**(3), 931–959 (Sep 2024). https://doi.org/10.1137/23M1602103
8. Cooper, K.D., Harvey, T.J., Kennedy, K.: A simple, fast dominance algorithm. Technical Report TR-06-33870, Rice University (2001), https://repository.rice.edu/items/99a574c3-90fe-4a00-adf9-ce73a21df2ed
9. Cui, S., Zhao, G., Gao, Y., Tavu, T., Huang, J.: VRust: Automated Vulnerability Detection for Solana Smart Contracts. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 639–652. ACM, Los Angeles CA USA (Nov 2022). https://doi.org/10.1145/3548606.3560552
10. DODO: DODO: A Next-Generation On-Chain Liquidity Provider Powered by Pro-active Market Maker Algorithm. Whitepaper, DODO (2020), https://dodoex.io/docs/docs/whitepaper
11. Easley, D., O'Hara, M.: Price, trade size, and information in securities markets. Journal of Financial Economics **19**(1), 69–90 (1987). https://doi.org/10.1016/0304-405X(87)90029-8

12. Glosten, L.R., Milgrom, P.R.: Bid, ask and transaction prices in a specialist market with heterogeneously informed traders. Journal of Financial Economics **14**(1), 71–100 (1985). https://doi.org/10.1016/0304-405X(85)90044-3
13. Grech, N., Lagouvardos, S., Tsatiris, I., Smaragdakis, Y.: Elipmoc: Advanced decompilation of Ethereum smart contracts. Proceedings of the ACM on Programming Languages **6**(OOPSLA1), 1–27 (Apr 2022). https://doi.org/10.1145/3527321
14. Heimbach, L., Schertenleib, E., Wattenhofer, R.: Risks and Returns of Uniswap V3 Liquidity Providers. In: Proceedings of the 4th ACM Conference on Advances in Financial Technologies. pp. 89–101. ACM, Cambridge MA USA (Sep 2022). https://doi.org/10.1145/3558535.3559772
15. Helius: Solana's Proprietary AMM Revolution (Aug 2025), https://www.helius.dev/blog/solanas-proprietary-amm-revolution
16. Huang, R.D., Stoll, H.R.: The Components of the Bid-Ask Spread: A General Approach. Review of Financial Studies **10**(4), 995–1034 (Oct 1997). https://doi.org/10.1093/rfs/10.4.995
17. Jupiter: Jupiter Developer Documentation (2026), https://dev.jup.ag/docs/get-started
18. Kang, M.G., McCamant, S., Poosankam, P., Song, D.: DTA++: Dynamic Taint Analysis with Targeted Control-Flow Propagation. In: Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS). Internet Society, San Diego, CA (2011)
19. Kyle, A.S.: Continuous auctions and insider trading. Econometrica **53**(6), 1315–1335 (1985). https://doi.org/10.2307/1913210
20. Lehar, A., Parlour, C.: Decentralized Exchange: The Uniswap Automated Market Maker. The Journal of Finance **80**(1), 321–374 (Feb 2025). https://doi.org/10.1111/jofi.13405
21. Lifinity: Lifinity: Proactive Market Making with Concentrated Liquidity. Whitepaper, Lifinity (2022), https://lifinity.io
22. Milionis, J., Moallemi, C.C., Roughgarden, T., Zhang, A.L.: Automated Market Making and Loss-Versus-Rebalancing (May 2024). https://doi.org/10.48550/arXiv.2208.06046
23. Smolka, S., Giesen, J.R., Winkler, P., Draissi, O., Davi, L., Karame, G., Pohl, K.: Fuzz on the Beach: Fuzzing Solana Smart Contracts. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 1197–1211. ACM, Copenhagen Denmark (Nov 2023). https://doi.org/10.1145/3576915.3623178
24. Udeshi, M., Krishnamurthy, P., Pearce, H., Karri, R., Khorrami, F.: REMaQE: Reverse Engineering Math Equations from Executables. ACM Transactions on Cyber-Physical Systems **8**(4), 1–25 (Oct 2024). https://doi.org/10.1145/3699674
25. Weiser, M.: Program Slicing. IEEE Transactions on Software Engineering **SE-10**(4), 352–357 (Jul 1984). https://doi.org/10.1109/TSE.1984.5010248
26. Xu, J., Paruch, K., Cousaert, S., Feng, Y.: SoK: Decentralized Exchanges (DEX) with Automated Market Maker (AMM) Protocols. ACM Computing Surveys **55**(11), 1–50 (2023). https://doi.org/10.1145/3570639
27. Yuan, S., Zhang, Z., Lu, J., Sanan, D., Chang, R., Zhao, Y.: A Complete Formal Semantics of eBPF Instruction Set Architecture for Solana. Proceedings of the ACM on Programming Languages **9**(OOPSLA1), 1–27 (Apr 2025). https://doi.org/10.1145/3720414