pnl: An Algo Virtual Machine for Perpetual Futures

tradepnl.xyz

Version 1.0 • Language 1.0.0
November 2026

Abstract

We describe **pnl**, a platform on which systematic trading strategies are authored as programs, admitted by a statistical gate, and published as immutable listings whose performance is measured by independent Copy and Counter paper runs against live order books. Users take signed positions in these listings from their own accounts at a perpetual DEX; the platform never custodies funds.

The design rests on one property: a strategy emits a *target position series*, and target position series superpose linearly. This supports portfolio netting and a linear decomposition of target exposure. Realized accounting follows reconciled execution records.

We use the vocabulary of a layer-one blockchain throughout, because the correspondence is structural rather than decorative — programs compile to bytecode with declared resource bounds, must pass a reproducible tradeability test on historical data, which we call Proof-of-Backtest, and are frozen at publication by a constraint the database enforces. Section 12 states precisely where the correspondence fails, because a metaphor that is not bounded is a claim that cannot be checked.

1 The correspondence

Ethereum	**pnl**
EVM	AVM — the algo virtual machine
Solidity source	the pnl strategy language, a typed Python subset
Contract bytecode at an address	private strategy bytecode committed by a `strategyHash`
Gas	`fuel`, its worst case proven at compile time
Storage	`keep` and `window`, declared in the manifest
Block	one one-minute bar
Transaction	an account's rebalance toward new targets
Deployment	a listing
Immutable once deployed	immutable once listed
Upgrade = new deployment	change = new listing
Pending transaction pool	leased computation job pool (bounded analogy)
Execution release	pinned `engineHash`

The platform is not a blockchain. Private team workers run reproducible jobs; router-service controls job leases and listing, and execution-node coordinates trading. There is no consensus or chain of strategy blocks. The useful correspondence is deterministic execution of immutable programs on a fixed clock, with declared resource bounds.

2 The algo virtual machine

2.1 The program model

An AVM program is a *manifest* and a *step*. The manifest declares everything the program may consume: its markets (at most 64), how often it runs (`every`, one of 1m, 5m, 15m, 1h, 4h or 1d, with an optional whole-minute offset; one minute by default), its warm-up depth (at most three years), its kept memory (`keep` and `window`, at most 262,144 slots together), and its tunable parameters. Its fuel is not declared: the compiler computes the worst case per evaluation, which must fit the cap of 2,000,000 units times the stride in minutes, at most fifteen times.

`step` runs once per bar of the stride, a bucket of one-minute bars aligned to UTC; durations count those bars. A backtest may simulate on the buckets for speed, but evaluation, paper and live hold each decision minute by minute. It reads the declared features, reads and writes its own state, and emits a target position for each market it holds. Let $\mathcal{M}$ be the declared markets, and $\mathcal{H} \subseteq \mathcal{M}$ the markets emitted into. A program is a map

$$\Phi : \mathbb{R}^{|\mathcal{M}|\times k} \times \mathbb{R}^{s} \longrightarrow \mathbb{R}^{|\mathcal{H}|} \times \mathbb{R}^{s}$$

which is *pure*: identical inputs give identical outputs. There is no wall clock, no hidden randomness and no I/O inside `step`. A program that compiles is guaranteed both backtestable and runnable, because the same bytecode drives the backtest and the live runner.

2.2 Arrays, memory and randomness

A value is either a scalar or an array of float64 — a vector `vec[n]` or a matrix `mat[r, c]` — whose shape is fixed at compile time, so every array operation's fuel and memory are static and the worst case is still proven before the program runs. `close[:]` reads a channel across the declared markets, and a `time` channel reads the bar's verified UTC start, so calendar phases are arithmetic on data rather than a clock. Broadcasting is numpy's for rank 1 and 2, with `reshape` and slicing; reductions run in a fixed ascending order, and their `nan` forms skip missing values. Named memory is declared as `keep` (persistent scalars, vectors and matrices) and `window` (a ring of the last n rows, NaN until full); an optional `init` block runs once before the first bar and cannot read market data. The primitives are low-level compute in the numpy and BLAS sense: `@`, `transpose`, `outer`, fixed-order `cholesky`, `cho_solve`, `solve_triangular`, rank-1 `cholupdate` and `choldowndate`, and cyclic-Jacobi `eigh` for $n \leq 128$; `argsort`, `take` and bit sets (`bitwise_xor`, `bit_length`); elementwise functions over the engine's own math, including the usual activations; and a stateless counter-based generator (`uniform` and `normal` of $(seed, i)$, a pure function of its arguments). Models — principal components, kernel and Gaussian-process regression, filters, persistent homology — are programs written from these, so their cost is the sum of priced instructions.

A script is *inference, not training*. The author writes the model's evaluation and declares its searchable constants as `param`s with bounds; the platform's search over those bounds is the training step, and every trial it runs is counted by the gate. Updates a model makes as it runs (a Kalman filter, recursive least squares) are walk-forward inference. Every parameter element and every array literal in the body counts as a tunable coordinate (`tunableScalars`) for the gate's trial accounting. Scripts are the only strategy form.

2.3 Missing data and explicit populations

A missing bar reads as NaN. Plain reductions (sum, mean, median, minimum, maximum, population standard deviation) propagate it, and their `nan` forms skip it; a population is selected with `where` over a mask, so which members count is written in the program. The real-valued program model above is therefore extended with explicit missing values at the numerical boundary.

2.4 Fuel bounds and runtime accounting

The compiler computes the worst-case work of one bar and refuses programs whose worst case exceeds the per-bar cap B. The VM charges the same instruction prices at runtime, so compilation and metering enforce one resource contract:

$$F_{\max}(P) \;\le\; B \qquad \text{checked once, at compile time.}$$

Loop trip counts must therefore be statically known. Runtime enforcement remains a defensive boundary: exceeding the worst case is a fault, not permission to run partial trading logic. A run also has a budget in fuel, so where a long backtest stops is deterministic rather than a timing accident. The same engine and data bindings are required for a reproducible backtest.

3 Blocks

The platform's clock is a one-minute bar (60 seconds). Every listed program's `step` is evaluated once per bar against the bar that has just closed, producing a new vector of target positions for every live listing. Those targets are broadcast to every account holding the listing; each account computes the difference between where it is and where its holdings now say it should be, and that difference is its transaction for the block.

Two consequences follow, and they are the practical payoff of the design.

Execution is stateless. An account's required trade is a pure function of its current position and the current targets. There is no per-user, per-algo state machine to advance, no subscription lifecycle, and nothing to resynchronise after a restart.

A point for bar t requires bar $t+1$. The return earned by a position held through an interval is known only once the interval has closed, so the record is always one bar behind the clock. This is causality, not latency.

4 Positions superpose

4.1 The holding

An account u holds listing i with a fraction $f_{u,i} \in [0,1]$ of its margin and a signed multiplier $m_{u,i}$. A negative multiplier is a *counter*: the mirrored target sequence, a real position rather than a metaphor, and the means by which a user expresses that an edge is crowded without needing its author to agree.

Writing $s_i(k,t)$ for listing i's target in market k as a fraction of its normalized portfolio exposure, and $E_u(t)$ for the account's sizing capital, the account's target notional is

$$n_u(k,t) \;=\; E_u(t) \sum_i f_{u,i}\, m_{u,i}\, s_i(k,t). \tag{1}$$

Targets share one gross exposure budget:

$$s_i(k,t) = \frac{h_i(k,t)}{\max(1, \sum_j |h_i(j,t)|)}, \qquad \sum_k |s_i(k,t)| \le 1.$$

The maximum total target notional for this allocation is $E_u f_{u,i} |m_{u,i}|$. Three fully active, equal-weight markets share that budget equally; inactive markets do not reserve an equal slice. Long and short notionals both count toward the gross limit. There is no additional leverage factor inside the strategy. Price movement and execution can cause observed positions to differ from requested targets.

Exchange leverage and margin mode belong to the account and market, where several algos may contribute to the same net position. Cross-margin positions share account collateral; isolated-only markets retain separate collateral. An algo allocation is a sizing budget, not a separately locked margin balance.

4.2 Dedicated perpetual-account management

After accepting the account-management disclosure, the user's first allocation authorizes management of the entire linked perpetual account. The target is the combined algorithm portfolio; markets outside it target zero. Existing perpetual orders are reconciled and manual orders cancelled. Users keep manual trading in a separate account. Spot trades and asset transfers are outside this authority.

Before placing its first order, the execution service records an immutable observation of existing inventory under the account's ownership lease. This is not a synthetic fill and creates no historical algorithm performance or points. Subsequent inventory must reconcile with that observation and confirmed fills; unexplained external activity is not silently imported. Initial adoption blocks trading until complete. For an adopted account, reconciliation failures block increases while venue-enforced reductions remain possible. Closing existing positions can realize losses and incur fees.

4.3 Account protection constrains exposure

The user chooses an adverse-move buffer for the account. Increasing this buffer reduces the admissible leverage; changing it does not silently resize an allocation, rewrite a ticket draft, or close a position. The basket's un-netted sizing bound is

$$G_u = \sum_i f_{u,i}|m_{u,i}| \le L_u^{\text{protection}},$$

where the right-hand side is computed from the chosen buffer, venue margin tiers and trading costs rather than a universal fixed multiple. Copy and Counter share the more restrictive directional limit. Displayed limits allow for concentration in any traded market, without relying on equal weights or cross-market hedges.

For current notional magnitude N and buffer b, a short's adverse price move raises its stressed magnitude to $N(1+b)$, whereas a long's decreases it to $N(1-b)$. The protection model reserves adverse P&L, tiered maintenance margin and trading fees. Pending orders are assessed through fill and no-fill extremes; an unfilled reduction does not supply additional risk capacity. Cross positions share a collateral bucket, while existing isolated positions are assessed against their own equity.

Execution reads the complete account before increasing exposure, including positions and orders placed outside pnl. Initial-margin capacity is checked separately from the stress requirement. Venue leverage may be configured using existing collateral, but the new setting requires observed confirmation and a fresh risk check before an order is sent. Venue leverage controls required margin; it is not an additional multiplier on (1). Ownership, authorization and protection checks also run at the submission boundary. Missing or inconsistent observations block increases. Venue-enforced reduce-only exits remain available, and pnl's increasing working orders may be cancelled when protection is no longer sufficient.

The current automatic-increase path supports cross margin. Hyperliquid unified USDC accounts require proof that other perpetual DEXes and spot orders do not introduce unmodelled exposure; portfolio margin and discontinued DEX abstraction are refused. Lighter unified accounts require complete evidence that no non-USDC assets are used as margin. Existing isolated exposure is included in risk assessment, but isolated increases and isolated-only openings

are refused until an isolated collateral workflow is supported; they are never silently converted to cross margin.

These checks bound permitted actions under a specified model, not future loss. Even 1× allocation leverage is not a liquidation guarantee: gaps, funding, fees and changing account state can exceed model assumptions. Liquidation remains governed by the venue.

4.4 Netting is not an optimisation

Because (1) is a sum, two listings that disagree about k contribute terms of opposite sign and the account trades the difference. The exchange account holds a net position per market. Per-listing contributions describe the intended decomposition; actual positions follow accepted orders and reconciled fills.

4.5 Attribution follows execution

Over interval t, with $r(k,t)$ the market return, φ the attributed trading cost and fund the attributed funding,

$$\Delta\mathrm{PnL}_{u,i}(t) = \sum_k n_{u,i}(k,t-1)\, r(k,t) - \varphi_{u,i}(t) - \mathrm{fund}_{u,i}(t), \tag{2}$$

This is the linear attribution model. Realized accounting must use reconciled fills, fees and funding, with explicit handling of rounding and unmatched activity. Target contributions alone do not prove that an order filled or that every account balance change belongs to a listing.

5 Proof-of-Backtest and certification

The gate asks whether a strategy's history is tradeable as a market, not whether it was profitable. Full evaluation works on daily net returns over the whole available history and passes when either of two paths holds: *drift*, where net returns after all fees are reliably positive, or *fee-aware persistence*, where pre-fee regimes persist long enough that following them beats the fees a holder pays to switch sides. Each path must hold in at least three of four quarters, and the history must span at least 420 days. Liquidation or insufficient evidence is a refusal. There is no profitability, readiness or approval-probability score.

Fast evaluation reports the same tradeability measures at a small resampling budget, with a fee-sensitivity fingerprint and parameter perturbations as diagnostics; it is feedback, not a verdict. Full evaluation runs on private backtest-node workers at full budget. router-service checks the result's strategy, engine, network, dataset and policy bindings before listing. Worker signatures bind origin and inputs; they do not independently prove correct computation. There is no secret historical holdout and no required out-of-sample or refitted walk-forward split. The design targets medium- and low-frequency strategies for which calendar history and independent trading episodes are limited.

The number of strategies an owner tried counts against a pass, so searching harder does not make passing easier; it cannot account for searches the platform never saw. Full evaluation returns a pass or refusal under the tradeability policy, identified by its digest, with failure feedback for the owner. Insufficient evidence is never promoted to a pass. The gate filters evidence; it does not prove a future edge. The launch gate page, `https://docs.tradepnl.xyz/concepts/gate/`, is the canonical statement of what full evaluation decides.

6 Architecture

Every service is operated by the team; users never run nodes, and there is no public dataset. router-service is the single API: authentication, every app database write, websockets, and three

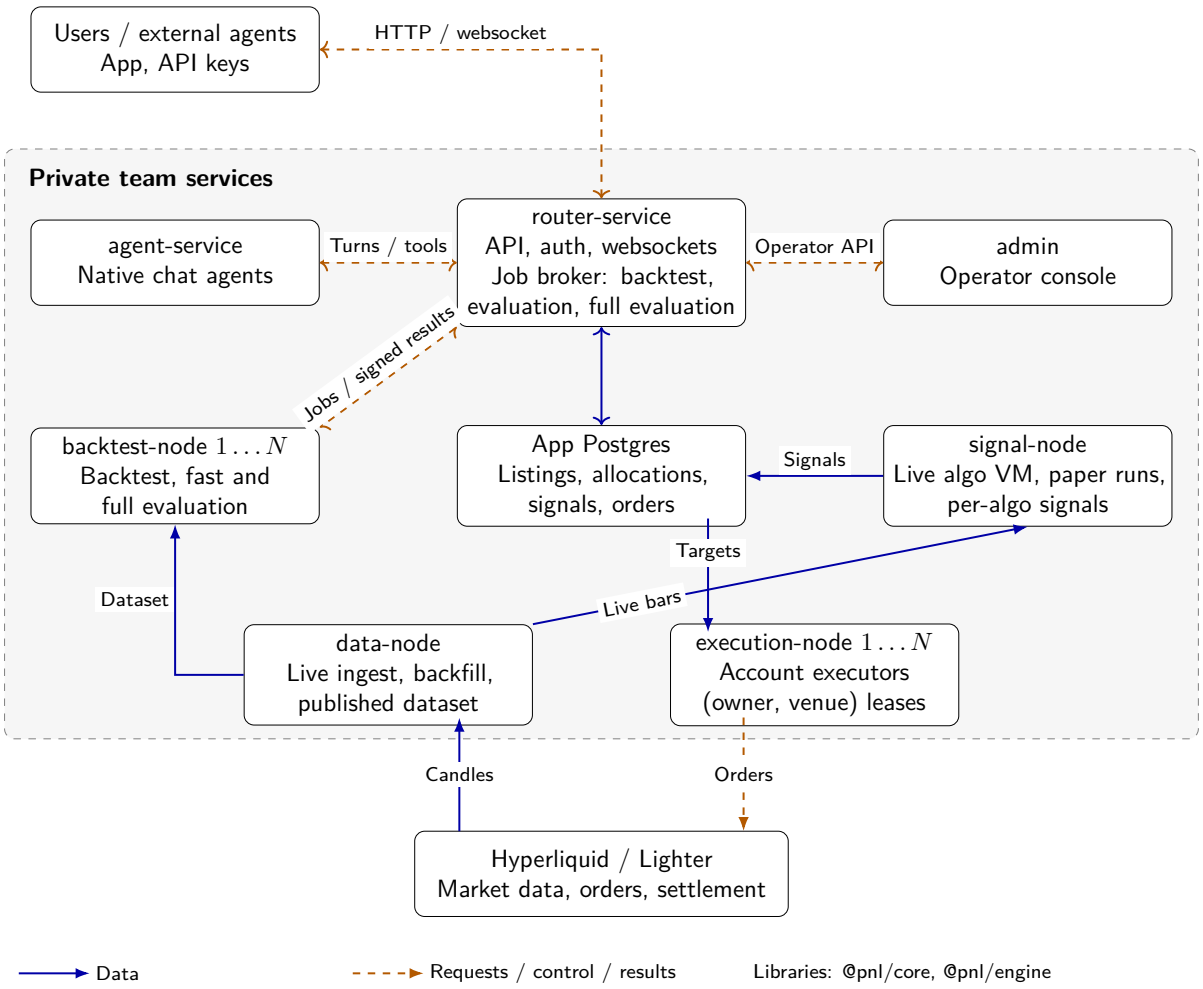


Figure 1: Team-operated services. Solid arrows carry data; dashed arrows carry requests, results and control.

bounded job queues for backtests, fast evaluation and full evaluation. backtest-node workers claim those jobs and return signed results. data-node ingests venue one-minute candles and funding, repairs history and publishes an immutable, hash-chained dataset that workers read directly. signal-node runs every listed algo live with its Copy and Counter paper runs and publishes per-algo signals. execution-node executors lease (owner, venue) groups, compute each account's target as the sum of allocation weight times algo signal, and trade it. agent-service runs native chat agents through router-service, and admin is the operator console. Shared code lives in two private libraries, `@pnl/core` and `@pnl/engine`; signal-node and backtest-node pin the same engine, so a backtest and a live listing run identical bytecode. Each service has an explicit mainnet/testnet mode.

Listed strategies stay on team infrastructure: scripts, tuned parameters and input markets are never distributed. Public listing responses expose performance summaries, curves and trading activity, not the strategy. Observed trading activity is public by design; this boundary does not promise that public behaviour cannot be analysed.

API keys authorize hosted backtests and fast evaluation; paid full evaluation and listing require the owner's authenticated wallet session, so an agent hands its selected strategy back to the owner. A passing full evaluation returns an owner-bound, expiring, one-use `verdictId`. Only afterward does the owner provide listing metadata to `POST /api/v1/algos`, which checks the verdict and strategy bindings and consumes it atomically.

Jobs bind the exact manifest sequence and digest, so archived replay never substitutes newer history. Missing data is NaN, never zero. Where a market's price is not observed for a period, it may be represented as unchanged (flat, no volume). A snapshot runs through its declared endpoint and may lag the newest closed bar.

One executor can hold every (owner, venue) group; more executors split them. They are one logical service, not validators. Independent execution operators with consensus over target positions remain future work and would require explicit membership, ordering, data agreement and fault rules.

7 Immutability

Once listed, a program cannot change. Frozen: the strategy hash, engine hash, execution parameters, bytecode and its surface digest, language version, markets, venue, dataset hash, gate verdict, drawdown, deployer and listing time. Mutable: lifecycle status, and presentation — name, ticker, description.

Enforcement is a `BEFORE UPDATE` trigger subtracting the mutable set from the whole row, so the rule is *deny by default*: a column added to that table next year is immutable the day it is added.

There is no upgradeable-proxy pattern and there will not be one. In Ethereum that pattern is the largest single source of "the code you audited is not the code running"; here there is no indirection between a listing and its bytecode, so there is no swap to perform and no owner who could perform it. `relistedFrom` records provenance and carries nothing else — no state, no record, no Proven badge.

This is what makes a track record mean anything. Forward evidence is measured from the listing time against a hash that cannot move; were the code editable, a curve would describe something that no longer exists, and the edit that mattered most would be the one made after the record looked good.

7.1 Forward evidence labels

Trial listings remain copyable after verification and the normal lifecycle checks. Proven requires an explicit, audited review of the live record and can be revoked by review. There is no automatic

graduation timer. A listing displays exactly one of Demo, Trial or Proven; examples display Demo instead of an evidence badge; testnet activity cannot establish mainnet performance.

8 The oracle

A listing has independently executed Copy and Counter paper records. Copy follows the strategy's target positions; Counter reverses those targets before execution. Each direction has its own quantities, book fills, fees and funding. Counter performance is not obtained by negating Copy returns.

For direction d, sizing capital is

$$C_d(t) = \max\left(\$1{,}000,\ \sum_{u \text{ allocated to } d} M_u^{\text{effective}}(t) a_u^{\text{effective}}(t)\right).$$

Here $M_u^{\text{effective}}$ is the holder's account equity and $a_u^{\text{effective}}$ is the allocation fraction from the same immutable effective boundary. Pending requests do not resize the existing basket. The multiplier's sign selects the direction; its magnitude is leverage and is not counted again as allocated capital. Paper evaluation uses a one-times gross exposure baseline shared across the traded markets. Different directional allocations can therefore incur different slippage.

Capital changes are cash flows, not returns. Carry and funding use the capital before resizing, and subsequent execution costs use the new capital. Persisted normalized interval returns are summed into the cumulative curve; the curve is not a compounded growth index. A liquidated direction stops trading and is not revived by a later allocation increase.

Each direction records performance from its actual start. Restart catch-up marks persisted positions without inventing historical book fills or applying today's allocations retroactively. Testnet activity does not establish mainnet evidence.

Current execution and finalized accounting advance separately. Strategies read verified closed one-minute bars; a fresh signal may inform current execution, whose fills walk the current book at the recorded execution time. Missing or stale depth makes the live extension unavailable rather than implying zero exposure or return. Missed execution opportunities are not replayed later.

Accounting finalizes when canonical prices and funding observations are available. The live extension remains provisional until then. Late funding corrections are journaled against their original minute and exposed through correction revisions; clients replace revised history rather than append a second return for that minute. Historical records retain their original methodology and provenance.

These paper records are not forecasts of an account. An account nets across its basket, has its own margin and entry times, and sends actual venue orders. Paper execution and live order management remain distinct fill mechanisms. Deployment and live acceptance of this methodology must be checked separately.

9 Fees

The platform fee is a fraction of executed notional set by the algo's turnover T: its average daily traded notional divided by its capital, a dimensionless rate per day in which opening and closing fills both count. Per fill,

$$f(T) = \min\Big(5,\ \max\big(0.1,\ \tfrac{5}{T}\big)\Big) \text{ basis points.}$$

Up to one turn of capital a day the fee is 5 basis points. Between $T = 1$ and $T = 50$ it is $5/T$, so the daily charge $f(T)\,T$ is a constant 5 basis points of capital: a fast strategy is not priced

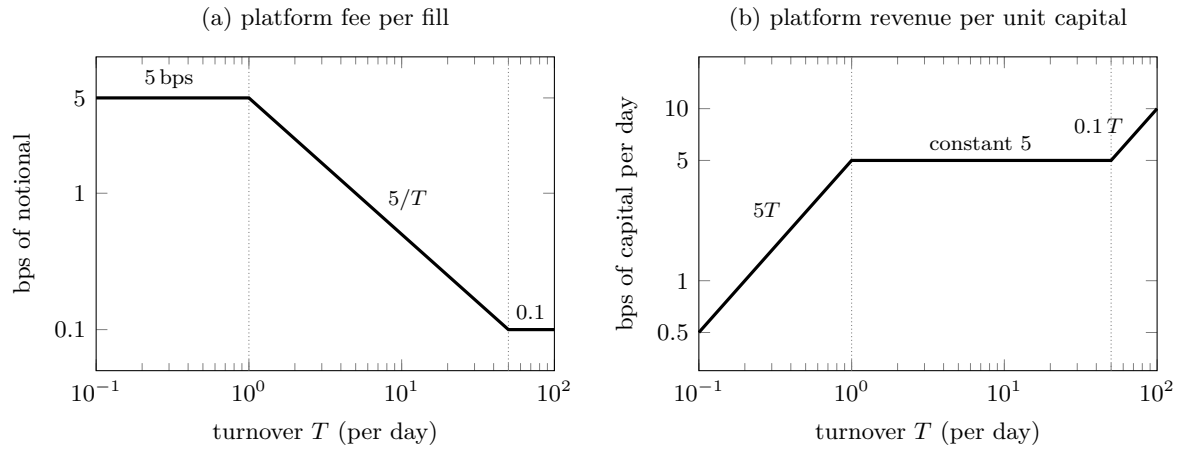


Figure 2: Schematic platform fee against turnover T (log scales). The fee is 5 basis points up to $T = 1$, $5/T$ across the constant-revenue band, and 0.1 basis points from $T = 50$. Revenue per unit capital, $f(T)\,T$, rises to 5 basis points a day at $T = 1$, holds there across the band, then rises as $0.1\,T$.

out by a fee sized for a slow one. Beyond $T = 50$ the fee stays at its 0.1 basis-point minimum and the daily charge grows again as $0.1\,T$ (Figure 2).

T is measured on the algo's published targets, so no wallet can change another's rate; recent activity counts more than old. Accounts net their algos first and the fee is charged on actual fills. Rates round down to the venue's fee unit (0.1 basis points on Hyperliquid, 0.01 on Lighter); a referral takes a separate 10% off, so a referred user pays at most 4.5 basis points. The signed approval ceiling is 5 basis points. Exchange fees and slippage are separate.

Backtests, evaluations and paper records charge this platform fee, without referral discounts, plus each venue's official base-tier fees: Hyperliquid tier-0 perpetuals at 1.5 basis points maker and 4.5 taker, Lighter standard accounts at zero. Every cost is a fraction of notional, so a backtest's returns do not depend on the capital it simulates. Fast evaluation reports a fee-sensitivity fingerprint: the backtest re-run with all trading costs scaled (for example 2× and 4×), described by how Sharpe ratio and return decay. It characterises the strategy; it is not a gate.

The split across listings is weighted by max(0, recent return) under a seven-day half-life. This split is deliberately non-linear: it stops a losing strategy being paid. A maximum is not a linear operator, so a user's realised cost depends on the whole basket rather than on each holding independently.

10 The ecosystem

The gate decides what may be listed. It does not explain why anybody would bother listing, and that question has a structural answer rather than a promotional one.

10.1 The asymmetry the platform exists to trade on

Writing a systematic strategy is hard and holding one is easy. Almost anyone who trades can decide that a published track record is worth a position; almost nobody can produce one. Today both sides of that asymmetry are stuck: the person who can build has no distribution beyond their own capital, and the person who cannot build has no access to anything except a screenshot and a Telegram invitation.

Nothing about this requires a platform in principle — anyone may write their own bot and run it. What they cannot do alone is be *held*. A strategy run privately earns only its author's

own P&L; the same strategy listed earns a share of the fees of everyone with exposure to it, and does so without the author selling anything, managing anybody's money, or making a claim they would have to stand behind.

10.2 The loop

1. A deployer searches for a program that passes Proof-of-Backtest and lists it.
2. Holders take signed positions from their own accounts.
3. Their trading pays the platform fee, and a share of it is attributed to the algos that caused the volume — weighted by recent return, so a losing algo is not paid.
4. That share is the deployer's payoff, and it scales with adoption rather than with a subsidy.
5. A larger pool of holders makes the search worth more, which funds more search, which produces more listings worth holding.

The reader who cannot build gets automation and a measured record to choose from. The reader who can build gets distribution. The reader who wants neither gets a public ranking of what is actually working right now, which is information that does not otherwise exist — a live, adversarially-gated cross-section of systematic strategies, with their costs attributed and their drawdowns stated.

10.3 What the loop does not rest on

This is the part worth stating plainly, because it is where the design differs from the thing it superficially resembles.

A token launchpad's flywheel is a price. Early participants are paid by later ones, the mechanism requires the asset to keep appreciating, and when it stops the loop does not slow down — it reverses.

This loop is paid out of trading fees on volume that would exist anyway. A deployer's revenue is a share of collected platform fees, at most 5 basis points of traded notional; it is bounded, denominated in the quote asset, and it does not require anybody to buy anything from anybody. A holder who closes their position stops generating fees and that is the end of it — there is no exit that has to be absorbed, because nothing was ever sold into.

That makes the loop slower than a token's and harder to start, which is an honest cost rather than a hidden one. It also makes it capable of running flat or backwards without collapsing, because the thing being accumulated is a record rather than a position in a queue.

10.4 Trust is the binding constraint, and it compounds

The reason to state all of this precisely is that adoption here is gated on trust rather than on yield, and trust is the one quantity the platform cannot manufacture. A track record earns it slowly and only by being real: measured forward, on a hash that cannot change, with the losses shown.

Every property argued elsewhere in this paper exists to make that accumulation possible. Immutability (§7) is what stops a record describing something that no longer exists. Explicit sizing, capital-flow accounting and methodology boundaries make records interpretable without concealing changes in exposure. The §12 section is here because a platform whose first published claim is found to be overstated does not get a second one.

11 The PNL token

PNL receives the platform's revenue. That is a contract rather than a policy, and the difference matters enough to be the organising idea of this section.

11.1 Fixed supply, no owner, permissionless burn

Supply is minted once, in the constructor, to a single address. There is no `owner`, no `mint`, no pause, no blocklist and no proxy — the same refusal of upgradeability that Section 7 makes for listings, applied to the token.

`burn` destroys only the caller's own balance. There is no `burnFrom`, so no privileged address can destroy a holder's balance to reshape the supply. `totalSupply` is therefore monotonically decreasing: it can fall and it can never rise.

11.2 Revenue buys it back and destroys it

Platform revenue arrives at a contract that swaps it for PNL and burns the result.

The property worth stating precisely: PNL can leave that contract in exactly one way, destroyed. There is no transfer of PNL, no sweep, and no owner who can move it. The buyback is not a promise to burn what was bought — the bytecode has no other exit. That is the whole reason it is a contract and not a treasury wallet with a stated policy.

11.3 What is trusted, and what it can cost

An honest account of a mechanism includes what it does not guarantee.

The operator chooses when to buy and at what slippage. It cannot choose to keep the PNL and it cannot take the revenue — neither has a withdrawal path. The worst a malicious operator can do is trade badly, burning less PNL per unit of revenue than it should, or do nothing at all and leave revenue idle. `minOut` is supplied per call as the defence against the first, because a fixed bound would be either useless or a permanent halt.

Revenue cannot be withdrawn, and that is a real tradeoff rather than a feature. If the router is ever wrong, deprecated or drained, revenue held at that contract is stranded permanently. The alternative is a rescue path, which is an address that can take the revenue and which holders would have to trust is never used. A stranded balance is a bounded and visible loss; a rescue key is an unbounded and invisible one. The mitigation is operational rather than structural: deploy against a router that is already liquid, and send revenue in increments rather than in one transfer.

11.4 What is not deployed

No PNL contract exists on mainnet. The token and the buyback contract are deployed on testnet and the mainnet configuration ships no token address at all — the application reads an unset address as null and hides every action that would need one, because a button that reverts against the zero address is worse than no button.

So everything in this section describes mechanism rather than operation. The contracts are written, tested and running against a testnet; none of it is live, and no revenue has been through any of it.

12 Where the analogy breaks

A bounded metaphor is checkable; an unbounded one is marketing.

Distributed computation is not consensus. Multiple workers can compute and replay the same inputs, but users still trust the published data, engine release and private evaluation policy. A signature authenticates a result; it does not make the system end-to-end trustless. Immutability makes unauthorized changes detectable under the enforced identity contract; it does not eliminate operator trust.

Blocks are not chained. Bar t does not commit to bar $t-1$. Strategy bars are not consensus blocks. Dataset manifests are hash-chained, a separate integrity mechanism which does not turn strategy execution into a blockchain.

Settlement is not ours. Positions live in the user's own account at Hyperliquid or Lighter. We hold a delegated trading key that cannot withdraw, enforced by the venue. In L1 terms we are closer to a block builder than to a settlement layer: we say what should be traded, and someone else's chain is where it happens.

Proof-of-Backtest proves less than Proof-of-Work. PoW is a statement about computation that is essentially impossible to fake. Proof-of-Backtest is a statistical test on historical data, and that data is finite, non-stationary and shared by every searcher. A gate filters evidence; it does not prove future behaviour. Repeated search can find attractive historical results even without a durable edge.

13 Open problems

Execution consensus (unimplemented). A trusted set of independent execution operators could eventually agree on deterministic algo positions. This is distinct from scaling execution-node across executors, which partition (owner, venue) groups and are not validators. Such a future protocol would need agreed data and engine versions, input ordering, finality, and explicit fault and membership rules. Position agreement would not imply agreement on external venue fills, latency or settlement effects. No such consensus protocol is implemented or claimed here.

Execution parity. The gate's model, the oracle's book-walk and the live executor are three different models. A single-listing account should reproduce its listing's oracle run including execution assumptions; it does not yet.

Composed history. Deep history for warm-up and backtesting may be composed from multiple sources for continuity. Venue-specific quantities such as funding may be unavailable (NaN) for parts of that history, so strategies should prefer relative measures over absolute levels.

Resource pricing. `fuel` and kept memory are bounded but not priced. Both are recurring per-bar platform costs for every live listing, and a bound is not a market.

14 Constants

The language and ABI below describe the pinned engine release, not a claim about every running deployment. Private policy thresholds are omitted.

Language version	1.0.0
Surface digest	`0x2e1293f9…`
Bar	60 s (1 minute)
Maximum markets	64
Maximum kept memory	262,144 slots
Maximum fuel per 1-minute evaluation	2,000,000
Maximum warm-up	1,576,800 minutes
Directional paper capital floor	$1,000
Platform fee per fill	0.1–5 bp

Architecture and operator procedures describe the release design; rollout status is separate. `https://docs.tradepnl.xyz`