Signals Protocol v1: CLMSR-based Daily Bitcoin Range Markets

Signals Research Team
jaegun@signals.wtf, danny@signals.wtf
Signals Protocol

December 22, 2025

Abstract

Prediction markets promise an efficient way to aggregate dispersed information into prices, but most on-chain designs inherit a structural limitation from P2P binary markets: as soon as outcomes become continuous, liquidity fragments across coarse strikes and users lose the ability to express precise views. Range partitions, constant-product AMMs, and ad-hoc orderbooks alleviate parts of the problem, but none provide full-range betting with unified liquidity, bounded maker loss, and an operationally tractable capital stack.

Signals Protocol addresses this gap by implementing daily BTC range markets on top of a single cost-function market maker, CLMSR (Continuous LMSR). The price axis $[L, U)$ is discretized into ticks via an `OutcomeSpec`, and one global potential $C(q) = \alpha \ln Z(q)$ prices all ticks and ranges simultaneously. Traders buy arbitrary contiguous price ranges as atomic claims; the **gross trading cashflow** is path-independent and fully determined by the start and end CLMSR states. The **settled maker P&L** additionally reflects the payout determined by the settlement tick, and maintains a closed-form worst-case loss bound $\leq \alpha \ln n$ (uniform prior) and exact range–binary equivalence.

Maker capital is packaged into a daily *Market-Cycle Batch* with an LP Vault. Each day's trading and settlement compresses CLMSR P&L and gross fees into a single scalar that is split across LPs, a Backstop Vault, and Treasury via a fixed Fee Waterfall. A batch mint/burn rule ensures that any two LP shares that underwrite the same sequence of daily markets earn exactly the same return, regardless of when deposits and withdrawals are requested within the day. On top of this, a Safety Layer ties depth to LP capital through entropy budgets, enforces a hard daily drawdown floor using Backstop Grants, and adds withdrawal lags plus a constrained LOLR-style role for Treasury. A Performance Layer then chooses depth, priors, and fee policy strictly inside this safe region to target a long-run Fee–Cost inequality while maintaining acceptable slippage for traders.

Finally, we describe an oracle and settlement architecture based on signed pull feeds with explicit timing windows, together with an implementation blueprint for an EVM rollup using immutable CLMSR math libraries, segment-tree-based distribution storage, and WAD-fixed arithmetic with explicit rounding invariants. The result is a solvency-aware, continuous-outcome prediction protocol currently operating daily BTC markets, with on-chain infrastructure that supports per-market oracle configuration for future asset expansion.

1 Introduction: Problem & Product

1.1 The P2P Binary Orderbook Model

Most on-chain prediction markets still rely on a **P2P (peer-to-peer) binary orderbook** model. The venue stays neutral by matching offsetting orders between participants so that directional risk is always transferred in a zero-sum fashion: one trader's loss is another's gain.

This structure works well for binary events with clearly defined YES/NO outcomes (e.g., "Will Bitcoin's price exceed $100,000 on December 31, 2024?"). Participants bet on one of two discrete outcomes, the engine pairs opposing orders on the same line, and the platform does not warehouse risk.

1.2 Structural Limitation on Continuous Outcomes

For **continuous variables** such as asset prices, the same P2P model runs into a structural limitation. The outcome space is a continuum; traders typically have views on specific ranges, not just on whether price ends up "above" or "below" a single strike.

Two effects appear:

- **Failure of coincidence of wants.** A trader may want to buy a narrow range such as \$60,100–\$60,200. Finding a counterparty who wants to sell that exact range, in the opposite direction, at the same time is unlikely. As the set of possible ranges grows, exact matching becomes harder.
- **Liquidity fragmentation.** To avoid this matching problem, venues usually list a small number of coarse strikes (e.g., "above \$50k", "above \$60k") and ignore the rest of the price axis. Liquidity is forced into these listed points, but information about the shape of the distribution between them is lost.

In practice, this means that P2P binary orderbooks do not scale well to full-range price discovery. They either fragment liquidity across many thin markets or compress information into a few wide buckets.

1.3 The Liquidity–Cost Trade-off (The CLMSR Solution)

Signals replaces the P2P matching model with a CLMSR-based automated market maker. Instead of searching for a counterparty for each range, a single cost-function maker takes the other side of all range trades.

This introduces an explicit trade-off:

- **Benefit.** A single CLMSR potential $C(q) = \alpha \ln Z(q)$ (Section 2) prices all ticks and ranges simultaneously. Traders can buy arbitrary contiguous price ranges as atomic claims, and liquidity is unified across the entire OutcomeSpec rather than fragmented across many orderbooks.
- **Cost.** The maker bears entropy cost and inventory risk. With a uniform prior, CLMSR's worst-case loss is bounded by $\alpha \ln n$ (Section 2). This bound increases with both depth α and resolution n; deeper and finer markets are more useful but more expensive to underwrite.

The design accepts this trade-off explicitly. Maker capital is packaged into an LP Vault (Section 3). A Safety Layer constrains worst-case loss and daily drawdown via entropy budgets, depth caps, and a Backstop Vault (Section 4). A Performance Layer then chooses depth, priors, and fee policy inside that region so that long-run fee flow is intended to cover entropy cost while maintaining usable slippage for traders (Section 5).

1.4 Maker Capital and the Role of LPs

A CLMSR maker must always stand ready to take the other side of range trades. Every time a trader opens (or increases) a range position R, the maker takes the other side of that claim and accumulates the corresponding settlement liability. Maintaining one global potential $C(q)$ with usable depth therefore requires **continuous maker capital** willing to bear entropy cost and inventory risk.

In principle, this capital could be provided entirely by a single protocol-owned pool. Signals instead packages the maker side into an LP Vault, allowing external LPs to supply capital and share in both risk and return. The reasons are primarily economic:

- **Depth scalability.** Daily CLMSR depth α must be large enough to keep slippage usable on the chosen OutcomeSpec. Solely protocol-owned capital would bottleneck depth and volume; sharing the maker role with external LPs lets depth scale with demand and capital inflow.
- **Explicit risk sharing.** By tokenizing maker capital as vault shares, we make the maker P&L path investable with a simple daily rule: each day's trading P&L, the portion of fees attributed

to LPs, and any tail-risk support from the Backstop are all reflected in the vault's net asset value before a single batch price is applied. Sections 3–4 formalize this accounting and the loss limits in precise terms. LPs opt in when they accept this risk–return profile under those constraints.

- **Separation of long-term insurance.** A dedicated Backstop Vault owns tail segments outside the LP design range, funded by a carved-out fee share, while Treasury plays a separate LOLR and operating-capital role (Section 4.2). This separation keeps the LP product economically clean while still allowing the protocol to manage systemic tail risk.

Concretely, LPs act as the CLMSR **equity tranche**: they primarily absorb entropy cost and routine volatility within the Safety Layer's design range ($\alpha_{\text{limit},t}$ and p_{dd}), and in return hold vault shares with a residual claim on the Fee Waterfall. Backstop truncates tails beyond that range by injecting capital on tail days when needed to respect the drawdown limit, and Treasury can intervene as LOLR in liquidity shocks. Section 3 formalizes how maker capital is packaged into a daily Market-Cycle Batch, and Section 4 specifies how the Safety Layer bounds the LP risk share.

1.5 Scope and Terminology

This document specifies the **protocol/operational layer** for Signals v1 (daily BTC range markets). Mechanism-level proofs and derivations live in a separate reference: *CLMSR Mechanism Whitepaper (2025)*.

Terminology

- **Fee policy**: per-trade / per-settlement charging rules that aggregate into a gross daily fee $F_{\text{tot},t}$.
- **Fee Waterfall**: the Safety-fixed priority mapping that splits fees and enforces solvency (Section 4.3).

2 CLMSR Pricing Engine

2.1 Theoretical Foundation and the Position of CLMSR

CLMSR is a structure that applies Hanson's LMSR to continuous outcomes by using **a single cost function discretized into tick units**. Since Section 1 established why we chose range-type markets and the necessity of a single potential function, here we clearly describe only the mathematical definitions and properties.

2.2 Outcome Space and Cost Function Definition

With **OutcomeSpec** (L, U, s, d), we divide $[L, U)$ into a uniform tick set $\mathcal{B} = \{0, \ldots, n-1\}$ where $n = (U-L)/s$. By design, OutcomeSpec is defined so that $U - L$ is divisible by s, making all tick widths equal. d is the number of fixed decimal places (decimals) for internal settlement input, determining the OutcomeUnit scale in which L, U, s, and the scaled settlement value x are expressed. In the deployed v1 implementation, `OracleModule` fixes the oracle settlement value scale at six decimals; operational BTC markets therefore use $d = 6$, and future asset markets requiring another OutcomeUnit scale would need a coordinated change in `_convertPriceToSettlementValue`. The raw oracle feed has its own per-market `feedDecimals`; the oracle module scales the raw feed price into OutcomeUnit before tick mapping. In v1, an implementation cap $n \le n_{\max}$ (currently $n_{\max} = 256$) is enforced for gas/DoS safety; OutcomeSpec templates must satisfy this bound.

Each daily market t ultimately resolves to a single tick index $\tau_t \in \mathcal{B}$ (the settlement tick) obtained from the oracle via OutcomeSpec's tick mapping (Section 6). A trading position in market t is a range claim (R, x) where $R \subseteq \mathcal{B}$ is a contiguous tick range and $x \ge 0$ is the position quantity (in settlement-token units). Its settlement payout is

$$\text{Payout}(R, x; \tau_t) := x \cdot \mathbf{1}[\tau_t \in R].$$

On-chain range representation (half-open boundaries). On-chain, a contiguous range claim is represented by boundary ticks $(\texttt{lowerTick}, \texttt{upperTick})$ and denotes the half-open set of bins

$$R(\texttt{lowerTick}, \texttt{upperTick}) := \{\, b \in \mathcal{B} \;:\; \texttt{lowerTick} \le b < \texttt{upperTick} \,\},$$

where $0 \le \texttt{lowerTick} < \texttt{upperTick} \le n$. Note that $\texttt{upperTick}$ is a one-past-the-end boundary and may equal n to include the last bin $(n-1)$. This convention removes ambiguity at boundaries and matches the discrete bin mapping used by the implementation.

Pricing state (softmax coordinates). The CLMSR *pricing state* is a vector $\mathbf{q} = (q_0, \ldots, q_{n-1}) \in \mathbb{R}^n$. It is a real-valued coordinate system for log-weights and may contain negative entries (e.g., when embedding a non-uniform prior). Define the strictly-positive weights

$$w_i(\mathbf{q}) := e^{q_i/\alpha} > 0,$$

and the partition sum $Z(\mathbf{q}) := \sum_i w_i(\mathbf{q})$. The cost function is

$$C(\mathbf{q}) = \alpha \ln Z(\mathbf{q}), \qquad Z(\mathbf{q}) = \sum_{i=0}^{n-1} e^{q_i/\alpha}, \tag{1}$$

where $\alpha > 0$ is both the depth and the loss unit. Marginal prices/probabilities are

$$p_i(\mathbf{q}) = \frac{\partial C}{\partial q_i} = \frac{e^{q_i/\alpha}}{\sum_k e^{q_k/\alpha}}. \tag{2}$$

The partition sum Z binds all ticks together, enforcing $\sum_i p_i = 1$ and $p(\mathcal{B}) = \sum_{b\in\mathcal{B}} p_b$. If there is only one tick, it becomes classical LMSR. With this single equation, the meaning of "price = probability" is simultaneously defined across the entire range. Because C is a state function, any sequence of trades moving $\mathbf{q}_{\text{initial}} \to \mathbf{q}_{\text{final}}$ has gross trading cashflow $C(\mathbf{q}_{\text{final}}) - C(\mathbf{q}_{\text{initial}})$ (see Section 2.4, Path Independence).

Range pricing. At state $\mathbf{q}$, define the range price mass

$$p(R; \mathbf{q}) := \sum_{b\in R} p_b(\mathbf{q}).$$

The fee-free cost of increasing the range position by x is the CLMSR cost difference between the updated and current states. Let $\mathbf{q}'$ satisfy $q'_b := q_b + x$ for $b \in R$ and $q'_b := q_b$ otherwise; then

$$\Delta C(R, x) := C(\mathbf{q}') - C(\mathbf{q}).$$

This is the range trade analyzed in Section 2.3.

2.3 Range-Binary Equivalence

Theorem 1 (Partition Reduction)**.** *Adding quantity x to any range $R \subset \mathcal{B}$ gives*

$$\Delta C(R, x) = \alpha \ln \frac{Z_{\overline{R}} + e^{x/\alpha} Z_R}{Z_{\overline{R}} + Z_R},$$

where $Z_R := \sum_{b\in R} e^{q_b/\alpha}$ and $\overline{R} := \mathcal{B} \setminus R$ denotes the complement of R, so $Z_{\overline{R}} := Z - Z_R$. This is exactly the same as the cost of buying x on one side in a binary LMSR that treats the event as only $\{R, \overline{R}\}$. If there is only one tick, it reduces to LMSR.

Interpretation. Applying the multiplier $e^{x/\alpha}$ to R changes the partition sum to the numerator above, and this is exactly the binary LMSR cost formula. Since range orders are always reduced to a single binary market $\{R, \overline{R}\}$, the entire range is atomically updated and normalized.

Proof/derivations. See *CLMSR Mechanism Whitepaper (Signals Research Team, 2025).*

2.4 Path Independence

Path independence of gross trading cashflow. Since C is a function of state only, the **gross trading cashflow**

$$\Delta C = C(q_{\text{final}}) - C(q_{\text{initial}})$$

is determined solely by the start/end states, regardless of what order of trades occurred. Traders cannot create additional profit by reordering trades, and makers can reason about a full day of trading using endpoint snapshots rather than trade-by-trade paths.

3 Market-Cycle Batch & LP Vault

Signals packages CLMSR maker capital into a daily **Market-Cycle Batch** with an LP Vault. Each daily BTC market t is one batch: traders move the CLMSR state from $q_{\text{start},t}$ to $q_{\text{end},t}$; the protocol computes maker P&L and fees for that day; and all LP shares that underwrote that market move together to a new basis price. Deposits and withdrawals are only executed once per batch, at that daily equity price.

The batch exists to formalize three simple facts:

- maker P&L for a day is only well defined after settlement,
- fees belong to the capital that actually carried that risk from the first trade to settlement,
- LP shares that experienced the same markets should see the same return, regardless of when their owner sent deposit or withdrawal requests.

We now specify the minimal vault state variables and the batch mint/burn rules that implement these entitlements.

3.1 Why the Market-Cycle Batch Exists

For a single-owner CLMSR maker, daily P&L is straightforward. Given CLMSR states

$$q_{\text{start},t} \to q_{\text{end},t},$$

the gross trade cashflow is

$$\Delta C_t := C(q_{\text{end},t}) - C(q_{\text{start},t}),$$

fully determined by the endpoints (Section 2). The settled maker P&L L_t that LP shares absorb is defined in Section 3.4, incorporating the settlement tick τ_t (path-independent conditional on τ_t).

Once multiple LPs share maker capital, a basic question must be answered:

> Which shares are entitled to which day's P&L and fees?

Signals takes a simple, economic stance:

- LPs **earn trading fees because they stand ready to settle traders at market close**. They supply collateral so the protocol can guarantee payouts to winning positions. That entitlement arises from underwriting the market, not from holding the token abstractly.
- Underwriting a daily market t means **being in the vault from the first trade of t until its settlement**. Only those shares should absorb L_t and receive the LP fee share F_t (the LP-attributed fee term produced by the Safety Layer's Fee Waterfall; see Section 4.3 for the formal definition). Shares that arrive after settlement see the new basis price but do not retro-claim fees from markets they never underwrote.
- Conversely, a share that was present while fees were being collected for market t must also absorb the corresponding settlement risk for that same market. There is no fee without the obligation to be settled.

Without a batch abstraction, a continuously minting/burning vault would face an unattractive choice:

- either let capital that arrives after most of the day's risk has already been realized "buy into" that day's P&L and fee stream, diluting earlier LPs, or
- track a separate exposure schedule for each deposit and withdrawal so that every unit of capital has its own list of markets it underwrote.

The first option violates fee entitlement; the second option rebuilds a complex clearing engine on top of a mechanism that was explicitly designed to be summarized by a single potential function $C(q)$.

The **Market-Cycle Batch** is the minimal structure that avoids both outcomes. Each daily market t is treated as one indivisible unit of maker risk and fee accrual:

- all shares that are already issued when t opens and are still present when t settles move together from basis price P_{t-1} to a new equity price P_t^{e},
- deposits and withdrawals affecting which future markets a share underwrites are only executed *after* that move, all at P_t^{e}.

Everything else in this section is accounting detail to implement that principle.

3.2 One-Day Timeline for LPs (summary)

A market-day t is processed as one *Market-Cycle Batch*. Economically:

- During trading, the vault's *accounting state* $(N_{t-1}, S_{t-1}, P_{t-1})$ is fixed for pricing and risk accounting; only the CLMSR state q and the gross fee meter $F_{\mathrm{tot},t}$ evolve. LP deposits/withdrawals are submitted as *requests* and may escrow assets/shares immediately, but are tracked as *pending* objects and are excluded from (N, S) for the purpose of computing P until batch finalization.
- After settlement, maker P&L is computed per Eq. (4) and Safety produces a single net vault credit Π_t.
- The batch-finalization transaction applies Π_t to form N_t^{pre} and fixes the single batch equity price P_t^{e} per Eq. (3). It then deterministically assigns eligible deposit/withdrawal requests to batch t and records their claimable conversion at P_t^{e}.
- After batch t is finalized, each request is settled via *per-request claims* (e.g., `claimDeposit`/`claimWithdraw`). These claims are pure transfers/mints/burns at the fixed batch price and do not change P_t^{e}.

The normative batch procedure (including ordering, rounding, and request/claim semantics) is specified in Section 3.5 and Appendix C.

3.3 Vault State and Notation

At the end of each batch t the LP Vault is summarized by a small set of state variables:

- N_t: net asset value (NAV) of the vault, including CLMSR positions marked at cost, accumulated fees, and any Backstop Grants received. **Importantly, the day's settlement payout is reserved (escrowed) at settlement finalization and treated as a deducted liability in the vault's accounting; subsequent `claimPayout()` calls release this reserved amount and do not change LP NAV.**
- S_t: total LP shares outstanding.
- $P_t := N_t/S_t$: basis price per share.
- P_t^{peak}: running peak of P_t over all past batches.

- $\mathrm{DD}_t := 1 - P_t/P_t^{\mathrm{peak}}$: drawdown from peak, used by the Safety Layer to damp depth and size Backstop Grants.

In v1, assets/shares escrowed for pending LP deposit/withdrawal requests are tracked separately and excluded from the accounting NAV N_t (and thus from P_t) until they are converted at a finalized batch price.

During trading in daily market t:

- $(N_{t-1}, S_{t-1}, P_{t-1})$ stays fixed from the vault's standpoint; only the CLMSR state q and the gross fee meter $F_{\mathrm{tot},t}$ evolve.
- Deposit and withdrawal requests are recorded in queues and assigned to future batches. Requests may escrow assets/shares immediately (e.g., assets transferred in, shares locked/burned), but they are accounted for as *pending* objects and are excluded from the vault's accounting state (N, S) used to compute P until batch finalization assigns a single price P_t^{e}.

After market t closes and Safety has applied its capital flows, the LP-facing net LP-vault credit Π_t for batch t is known (see Section 4.3 for its decomposition into CLMSR P&L, LP fees, and Backstop Grants). The **pre-batch** NAV and equity price that apply to the shares that underwrote market t are defined as

$$N_t^{\mathrm{pre}} := N_{t-1} + \Pi_t, \qquad P_t^{\mathrm{e}} := \frac{N_t^{\mathrm{pre}}}{S_{t-1}}. \tag{3}$$

3.4 Daily Maker P&L from CLMSR and Fees

This subsection connects CLMSR trading cashflows, settlement payouts, and the daily net vault credit Π_t used in LP-vault accounting.

Gross trading cashflow. For market-day t, trading moves the CLMSR pricing state from $q_{\mathrm{start},t}$ to $q_{\mathrm{end},t}$. The maker's gross trading cashflow is the CLMSR cost difference

$$\Delta C_t := C(q_{\mathrm{end},t}) - C(q_{\mathrm{start},t}),$$

which is path-independent and fully determined by the endpoints.

Settlement payout and exposure ledger. A position in market t is a range claim (R, x) with $x \geq 0$ that pays x iff the settlement tick τ_t lies in R. The total settlement payout owed by the maker is therefore

$$\mathrm{Payout}_t := \sum_{\text{open positions } j} x_j \cdot \mathbf{1}[\tau_t \in R_j].$$

On-chain, the protocol tracks this payout surface as an outstanding exposure vector

$$\mathbf{Q}_t = (Q_{t,0}, \ldots, Q_{t,n-1}) \in \mathbb{N}_0^n,$$

where $Q_{t,b}$ is the total settlement-token amount owed if $\tau_t = b$. Opening or increasing (R, x) adds x to $Q_{t,b}$ for all $b \in R$. Decreasing or closing a position subtracts over the same range and must leave $Q_{t,b} \geq 0$ for all b (positions are represented as non-negative quantities). In exact arithmetic, when each trade applies the same per-range $\pm x$ update to both representations, we have $Q_{t,b} = q_{\mathrm{end},t,b} - q_{\mathrm{start},t,b}$.

Settled maker P&L.

$$\Delta C_t := C(q_{\mathrm{end},t}) - C(q_{\mathrm{start},t}), \quad \mathrm{Payout}_t := Q_{t,\tau_t}, \quad L_t := \Delta C_t - \mathrm{Payout}_t. \tag{4}$$

At settlement finalization, the protocol reserves Payout_t in a payout escrow. From the LP Vault's perspective, this amount is treated as a liability and is therefore already incorporated in the day's

accounting when the batch equity price is formed. Subsequent `claimPayout()` calls simply release funds from this reserved amount; they do not change LP NAV or the batch price for that market-day.

Define realized entropy loss

$$L_t^- := \max(-L_t, 0).$$

Loss bounds and entropy-budget objects (including $\alpha_t \ln n$ under a uniform prior and the general-prior budget $E_{\text{ent}}(q_{0,t})$) are defined in the Safety Layer (Section 4).

Interface to Safety and vault accounting. The Fee policy aggregates per-trade and settlement fees into a gross daily fee $F_{\text{tot},t} \geq 0$ (Section 5). The Safety Layer maps raw market outputs $(L_t, F_{\text{tot},t})$ and the current capital-stack state into:

- a net LP-vault credit Π_t applied at batch finalization, and
- internal Backstop/Treasury flows and grants that maintain the Safety invariants.

By definition (Section 4.3),

$$\Pi_t = L_t + F_t + G_t,$$

where F_t is the LP-attributed fee term and $G_t \geq 0$ is the Backstop Grant used to enforce the daily drawdown floor. From the vault's standpoint, only Π_t enters the pre-batch NAV identity

$$N_t^{\text{pre}} - N_{t-1} = \Pi_t,$$

with the internal split into F_t and G_t treated as Safety-layer detail. This identity is the core accounting equation used throughout the document: Section 3.2 uses only the Π_t side when describing the vault's one-day timeline, while Section 4.3 specifies how Π_t decomposes into L_t, F_t, G_t.

3.5 Batch Mint/Burn Rules and Invariants

Once N_t^{pre} and $P_t^{\text{e}} = N_t^{\text{pre}}/S_{t-1}$ have been set for batch t, the vault must process all eligible withdrawals and deposits **without disturbing** this equity price. This subsection specifies the mint/burn rules that guarantee that property.

Price invariance under deposits and withdrawals. Deposits and withdrawals during batch t use price $P = P_t^{\text{e}}$. The update rules for deposits (add assets, mint shares) and withdrawals (burn shares, remove assets) both preserve

$$\frac{N'}{S'} = P,$$

so the basis price remains exactly P_t^{e} regardless of processing order. Full integer-rounding conventions are specified in Appendix C.

Batch finalization vs. per-request claiming. The Market-Cycle Batch is *economically* atomic in the sense that all conversions use a single equity price P_t^{e} and all shares that underwrote market t experience the same basis-price move from P_{t-1} to P_t^{e}.

However, the on-chain execution model in v1 is *request/claim*-based: batch finalization fixes P_t^{e} and records deterministic claimable entitlements for the set of requests assigned to batch t, while each request is later settled via an individual claim call. This avoids requiring an on-chain loop over all pending requests during batch finalization and preserves gas/DoS safety.

Batch algorithm (v1 normative semantics). Putting the pieces together, the Market-Cycle Batch for day t is executed in two phases:

1. **Use finalized settlement and compute raw P&L inputs**
   - Use the finalized `settlementValue` and settlement tick τ_t for market t (Section 6).

- Compute maker P&L per Section 3.4: $\Delta C_t := C(q_{\text{end},t}) - C(q_{\text{start},t})$, $\text{Payout}_t := Q_{t,\tau_t}$ (from the exposure ledger), $L_t := \Delta C_t - \text{Payout}_t$.
- Aggregate gross fees $F_{\text{tot},t}$ from trades and settlements.

2. **Apply Safety capital flows (Fee Waterfall + Grant rule)**
   - Run the Safety Layer's Fee Waterfall and grant rule (Sections 4.3, 4.6) on $(L_t, F_{\text{tot},t}, N_{t-1}, B_{t-1})$.
   - Record the resulting net LP-vault credit Π_t and the internal tranche flows $(F_t, G_t, F_{\text{BS},t}, F_{\text{TR},t})$.
   - Update the LP Vault pre-batch NAV:
   $$N_t^{\text{pre}} := N_{t-1} + \Pi_t,$$
   leaving the underwriting share base at S_{t-1}.
   - In parallel, update Backstop and Treasury NAVs using $(F_{\text{BS},t}, F_{\text{TR},t}, G_t)$ as specified in Section 4.

3. **Fix the equity price for the batch**
   $$P_t^{\text{e}} := \frac{N_t^{\text{pre}}}{S_{t-1}}.$$
   This is the only price used for all deposit/withdraw conversions assigned to batch t.

4. **Finalize batch t and make queued requests claimable at P_t^{e}**
   - Determine the set of withdrawal requests whose lag has expired and that are assigned to batch t (Section 3.6). Each such request of x shares becomes claimable for
   $$A_{\text{wd}} := \left\lfloor x \cdot \widetilde{P}_t \right\rfloor$$
   settlement-token units at claim time, where $\widetilde{P}_t$ is the on-chain fixed-point representation of P_t^{e} (Appendix C).
   - Determine the set of deposit requests assigned to batch t. Each deposit of amount A becomes claimable for
   $$S_{\text{mint}} := \left\lfloor \frac{A}{\widetilde{P}_t} \right\rfloor$$
   LP shares plus a refundable residual
   $$A_{\text{refund}} := A - S_{\text{mint}} \cdot \widetilde{P}_t \;\geq\; 0.$$
   - Batch finalization records $(N_t^{\text{pre}}, P_t^{\text{e}})$ and the batch assignment for each request; it does not require iterating and settling all requests on-chain within the same transaction.

5. **Per-request claims (off the critical path)**
   - After batch t is finalized, any request owner may call claim functions to receive the assets/shares (and any refund) associated with their request.
   - Claims do not change the batch equity price P_t^{e} and do not retroactively change which shares underwrote market t.

6. **Record end-of-batch accounting state**
   - Record $P_t := P_t^{\text{e}}$ as the end-of-batch basis price used for drawdown tracking and for configuring the next market's Safety caps.
   - Update the running peak P_t^{peak} and drawdown DD_t used by the Safety Layer.

By construction, all shares that underwrote the same set of markets see the same basis-price path, and request timing within the day cannot be used to change exposure to market t.

3.6 Withdrawal Lag and Which Markets You Underwrite

For LPs it is natural to want flexible exits. For a CLMSR-based maker vault, however, *instant* redemption at current basis price would let LPs try to escape anticipated losses while still enjoying already accrued fees, which contradicts the underwriting principle.

Signals therefore ties exits explicitly to the batch structure:

- Any LP share that is present at the start of daily market t and still present when the batch for t runs must fully absorb that day's (L_t, F_t, G_t).
- A withdrawal request only affects **which future markets** the share will underwrite; it does not erase exposure to markets that are already in progress when the request is lodged.

To enforce this and to slow collective exits, Signals v1 uses a fixed **withdrawal lag** D_{lag}:

- A withdrawal request at time T_{req} becomes eligible in the first batch t' whose cycle time $T_{\text{cycle},t'}$ satisfies
$$T_{\text{cycle},t'} \geq T_{\text{req}} + D_{\text{lag}}.$$
- The request becomes *claimable* in the first batch t' whose cycle time $T_{\text{cycle},t'}$ satisfies $T_{\text{cycle},t'} \geq T_{\text{req}}+D_{\text{lag}}$. The claim amount is determined at the batch equity price $P^{\text{e}}_{t'}$ after applying $(L_{t'}, F_{t'}, G_{t'})$ for market t'. There is no lock on the basis price at request time.

Consequences:

1. **No "run ahead of the loss" option**
   An LP cannot observe an unfavorable CLMSR state or oracle movement, submit a withdrawal, and be guaranteed to exit at today's price before tomorrow's loss is realized. They remain exposed to the P&L of all markets whose batches occur before their request clears, including any Backstop Grants.
2. **Controlled outflow speed and reaction time**
   Even if many LPs request withdrawals after a drawdown, capital leaves the vault over at least a D_{lag}-sized window. The protocol has time to:
   - compress depth α_t and adjust fee policy,
   - consider Treasury LOLR interventions (Section 4.5),
   - and de-risk or pause markets if needed under governance,

   before depth collapses.

From the accounting perspective, withdrawal lag ensures that every executed withdrawal is cleanly tied to a single batch price $P^e_{t'}$, preserving the invariant that **all shares that underwrote the same list of markets see the same basis-price path.**

3.7 Token Representation and Secondary Liquidity

LP equity and trading positions carry different rights and lifecycles, so Signals represents them as different token types.

LP Vault shares (LP share token with batched accounting).

- LP shares represent pro-rata claims on the vault's accounting NAV and are freely transferable.
- The economic share value is the basis price $P_t := N_t/S_t$, which is fixed once per batch at P^{e}_t for the shares that underwrote market t.

- Mutating entrypoints for LP flows are *asynchronous request/claim*: users submit deposit/withdraw requests during the day, and once the assigned batch is finalized at price P_t^{e}, each request is settled via a per-request claim (shares minted/transferred, assets paid, and any refund returned).
- Secondary-market prices of LP shares are not enforced by the protocol; the protocol only fixes the on-chain basis price path via daily batches.

Trading positions (ERC-721).

- When a trader buys range $R \subseteq \mathcal{B}$ with quantity x in market t, the protocol issues a position NFT with metadata at least
$$(\text{marketId},\ R,\ x).$$
- This NFT represents a single eventual payout determined by R, the settlement tick for that market, and the CLMSR engine.
- Position NFTs are freely transferable. A buyer of a position NFT steps into the claim on the underlying range; the vault's maker economics are unchanged.

A dedicated settlement entrypoint (e.g., `claimPayout(positionId)`) checks that the associated market is finalized, computes the payout from R and settlementTick, pays it once, and marks the NFT as settled.

This separation keeps:

- **maker equity** composable via LP shares (structured products, vault-of-vaults, etc.), and
- **event outcomes** composable via NFTs (collateral, structured bets),

without entangling batch accounting with application-level use of those positions.

3.8 Rounding and Dust (summary)

All accounting is ultimately performed in integer settlement-token units. Canonical rounding and dust ownership rules (trade-side rounding, deposit/withdraw conversion, fee-split rounding, and grant rounding) are specified in Appendix C. This subsection summarizes the intent at a high level.

3.9 Interface to the Safety Layer

For each market-day t, Safety consumes $(L_t, F_{\mathrm{tot},t})$ and capital-stack state to produce (Π_t, B_t, T_t). The canonical Safety mapping (Fee Waterfall + grant rule) is specified in Section 4.

4 Safety Layer: Solvency and Invariants

The Safety Layer is the part of the protocol whose only job is "the system does not die, no matter what path trading takes." It defines hard constraints on top of the CLMSR engine (Section 2) and the daily Market-Cycle Batch (Section 3). On any admissible path and for every daily market t, it guarantees:

- **Depth tied to LP capital.** The depth used satisfies $\alpha_t \le \alpha_{\mathrm{limit},t} \le \alpha_{\mathrm{base},t}$, where $\alpha_{\mathrm{base},t}$ is derived from a single risk-budget parameter $\lambda \in (0,1)$ such that under a uniform prior the worst-case entropy loss consumes at most a fraction λ of LP capital.
- **Daily basis-price floor.** Let P_{t-1} be yesterday's LP basis price and P_t^{e} the pre-batch basis price after market t settles. Then
$$\frac{P_t^{\mathrm{e}}}{P_{t-1}} - 1 \ \ge\ p_{dd}, \qquad p_{dd} := -\lambda, \tag{5}$$
enforced, if necessary, by a Backstop Grant $G_t \ge 0$.

- **Backstop never negative on admissible priors.** Opening prior $q_{0,t}$ defines an entropy budget $E_{\text{ent}}(q_{0,t})$ split into a core and a tail ΔE_t. Only priors with $\Delta E_t \le B_{t-1}^{\text{eff}} \le B_{t-1}$ are admissible, and the grant rule ensures $0 \le G_t \le \Delta E_t$, so

$$B_t = B_{t-1} + F_{\text{BS},t} - G_t \;\ge\; 0$$

  on all admissible paths.

- **Same markets, same basis path.** Deposits and withdrawals are executed only once per day at the batch price P_t^{e}; withdrawal lag ensures any share present for market t participates fully in that market's P&L before it can exit.

- **Liquidity shocks cannot instantly collapse depth.** A fixed withdrawal lag D_{lag} slows outflows, and Treasury acts as a constrained Lender of Last Resort (LOLR) by buying LP shares in stress to restore depth capacity, never by paying Backstop Grants.

Sections 4.1–4.8 specify the state, parameters, and algorithms that implement these invariants.

4.1 Entropy Budget and the Single Risk Parameter λ

CLMSR uses

$$C(q) = \alpha \ln Z(q), \qquad Z(q) = \sum_b e^{q_b/\alpha}, \qquad p_b(q) = \frac{\partial C}{\partial q_b}.$$

For market-day t, the gross trading cashflow is $\Delta C_t := C(q_{\text{end},t}) - C(q_{\text{start},t})$ and the settled maker P&L is

$$L_t = \Delta C_t - \text{Payout}_t,$$

where Payout_t is the total settlement-token payout owed at the realized settlement tick τ_t (Section 3.4).

From the uniform state $q = 0$ on n ticks, $C(0) = \alpha \ln n$ and $p_b(0) = 1/n$. In the extreme where all mass collapses to one tick, $C(q) \to 0$, so a maker starting from $q = 0$ suffers at most

$$L_{\max}(0) \le \alpha \ln n.$$

This bound holds under the definition $L_t = \Delta C_t - \text{Payout}_t$ as well: by the log-sum-exp property, $C(q) \ge q_\tau$ always, so $C(q_{\text{end},t}) - q_{\text{end},t,\tau_t} \ge 0$ and thus

$$-L_t \le C(q_{\text{start},t}) - q_{\text{start},t,\tau_t} \le C(q_{\text{start},t}) - \min_j q_{\text{start},t,j}.$$

For a general opening state $q_{0,t}$,

$$L_{\max}(q_{0,t}) \le C(q_{0,t}) - \min_j q_{0,t,j}.$$

Entropy budget and tail split. Define the conservative worst-case entropy-loss budget:

$$E_{\text{ent}}(q_{0,t}) := \begin{cases} \alpha_t \ln n, & q_{0,t} = 0 \text{ (uniform prior)}, \\ C(q_{0,t}) - \min_j q_{0,t,j}, & \text{general prior.} \end{cases}$$

For fixed α_t, separate core and tail:

$$\Delta E_t := E_{\text{ent}}(q_{0,t}) - \alpha_t \ln n \;\ge\; 0,$$

so $E_{\text{ent}}(q_{0,t}) = \alpha_t \ln n + \Delta E_t$. The non-negativity $\Delta E_t \ge 0$ follows from the log-sum-exp inequality: for any $q_{0,t}$, $C(q_{0,t}) \ge \max_j q_{0,t,j} \ge \min_j q_{0,t,j}$, hence $E_{\text{ent}}(q_{0,t}) = C(q_{0,t}) - \min_j q_{0,t,j} \ge 0$. When $q_{0,t} = 0$ (uniform), $E_{\text{ent}}(0) = \alpha_t \ln n$ and $\Delta E_t = 0$. Concentrated priors with $\min_j q_{0,t,j} < 0$ increase ΔE_t proportionally, requiring additional Backstop coverage.

Risk budget and static core. Let

$$E_t := N_{t-1}$$

be LP capital at the start of market t. Choose a single risk-budget parameter $\lambda \in (0,1)$ meaning "a worst-case uniform-prior day may burn at most λ of LP capital." We define the static cap

$$\alpha_{\text{base},t} := \frac{\lambda E_t}{\ln n},$$

so that any admissible depth must satisfy $\alpha_t \le \alpha_{\text{base},t}$ and therefore

$$\alpha_t \ln n \le \lambda E_t,$$

and set the drawdown floor

$$p_{dd} := -\lambda.$$

Even if Backstop were empty and only uniform priors were admissible, LPs would lose at most λE_t in a day and face the floor p_{dd}.

Prior admissibility. Let B_{t-1} be Backstop NAV at the end of batch $t-1$ and $B^{\text{eff}}_{t-1} \le B_{t-1}$ the portion reserved for tail coverage (v1 uses $B^{\text{eff}}_{t-1} = B_{t-1}$). A prior is admissible iff

$$\Delta E_t(q_{0,t}) \le B^{\text{eff}}_{t-1}.$$

If not, governance must flatten the prior, fall back to (near-)uniform, or recapitalize Backstop before opening market t. This condition, combined with the grant rule in Section 4.6, ensures Backstop is never asked to fund more tail risk than it has reserved.

4.2 Capital Stack and Tail Allocation

The entropy budget is split across a three-layer stack:

Tranche	Role	Loss segment	Source of return
LP Vault	Maker equity	Core entropy/volatility	LP fee residual F_t
Backstop Vault	Tail insurance	Tail ΔE_t beyond core	Backstop fee $F_{\text{BS},t}$
Treasury	LOLR & ops	Liquidity/ops shocks	Protocol revenue / policy

LP Vault (equity). NAV N_t aggregates CLMSR positions, fees, and Backstop Grants; share value $P_t := N_t/S_t$. LPs absorb the core segment within $\alpha_{\text{limit},t}$ and p_{dd} and receive $F_t := F_{\text{loss},t} + F_{\text{LP},t}$ (dust already absorbed) in the accounting identity $N^{\text{pre}}_t - N_{t-1} = L_t + F_t + G_t$. If LP-attributed fee inflow covers realized CLMSR losses over time (Section 5.1), basis price P_t tends to grow structurally.

Backstop Vault (mezzanine). Above LPs, it owns the incremental tail ΔE_t:

- Normal days: accumulates fee share $F_{\text{BS},t}$ (Section 4.3), carved out from gross fees.
- Tail days: pays Grant $G_t \ge 0$ when raw LP return would breach p_{dd}, recording $B_t - B_{t-1} = F_{\text{BS},t} - G_t$.
- Priors are admissible only if $\Delta E_t \le B^{\text{eff}}_{t-1}$; under this condition the grant rule never demands more than ΔE_t of Backstop capital.

Treasury (LOLR & ops). Treasury buys LP shares at batch price in stress to restore depth when withdrawals/losses would push α_t above $\alpha_{\text{limit},t}$ (Sections 3.6, 4.5). It funds audits/upgrades/incidents and may recapitalize Backstop under governance. Treasury never pays G_t directly; all grants are Backstop $\to$ LP.

4.3 Daily Fee Waterfall and Capital Flows

Gross fees $F_{\text{tot},t}$ and maker P&L L_t from market t map to daily fee shares, grants, and the net LP-vault credit Π_t via a fixed four-step priority that keeps the capital stack solvent. Starting balances are $(N_{t-1}, B_{t-1}, T_{t-1})$, and Safety parameters include $(p_{dd}, \rho_{\text{BS}}, \phi_{\text{LP}}, \phi_{\text{BS}}, \phi_{\text{TR}})$.

1. **Loss compensation.**
   Use fees to offset realized entropy loss $L_t^- := \max(-L_t, 0)$:

$$F_{\text{loss},t} := \min(F_{\text{tot},t}, L_t^-),$$
$$F_{\text{pool},t} := F_{\text{tot},t} - F_{\text{loss},t}.$$

   Define the raw post-loss NAV

$$N_t^{\text{raw}} := N_{t-1} + L_t + F_{\text{loss},t}.$$

   This is the LP Vault NAV after CLMSR P&L and loss-offset fees but before any Backstop Grant.

2. **Drawdown floor and Backstop Grant.**
   Safety enforces the daily basis-price floor in Eq. (5) by computing a Backstop Grant $G_t \geq 0$ according to the canonical grant rule in Section 4.6. If the rule determines that the invariant cannot be satisfied on admissible priors (i.e., the required grant exceeds tail capacity), the batch reverts. Define intermediate balances:

$$N_t^{\text{grant}} := N_t^{\text{raw}} + G_t, \qquad B_t^{\text{grant}} := B_{t-1} - G_t.$$

   If $G_t > 0$, then necessarily $F_{\text{pool},t} = 0$ and no residual fee is distributed on that day.

3. **Backstop coverage target.**
   When $F_{\text{pool},t} > 0$ (i.e., there is residual fee after loss compensation and no grant), remaining fees push Backstop toward its coverage target. Given a target ratio and shortfall

$$B_t^{\text{target}} := \rho_{\text{BS}} \cdot N_t^{\text{grant}}, \qquad \Delta B_t^{\text{need}} := \max\{0,\, B_t^{\text{target}} - B_t^{\text{grant}}\},$$

   Safety dedicates part of $F_{\text{pool},t}$ to filling this gap:

$$F_{\text{fill},t} := \min(F_{\text{pool},t},\, \Delta B_t^{\text{need}}), \qquad F_{\text{remain},t} := F_{\text{pool},t} - F_{\text{fill},t}.$$

   If Backstop is under-capitalized, most of $F_{\text{pool},t}$ flows into $F_{\text{fill},t}$; if coverage is adequate, $F_{\text{fill},t} = 0$ and $F_{\text{remain},t} = F_{\text{pool},t}$.

4. **Residual distribution and rounding.**
   The residual fee $F_{\text{remain},t}$ is split using weights $(\phi_{\text{LP}}, \phi_{\text{BS}}, \phi_{\text{TR}})$ (summing to 1). On-chain we compute:

$$F_{\text{LP},t}^{\text{core}} := \lfloor F_{\text{remain},t} \cdot \phi_{\text{LP}} \rfloor,$$
$$F_{\text{BS},t}^{\text{core}} := \lfloor F_{\text{remain},t} \cdot \phi_{\text{BS}} \rfloor,$$
$$F_{\text{TR},t}^{\text{core}} := \lfloor F_{\text{remain},t} \cdot \phi_{\text{TR}} \rfloor,$$
$$F_{\text{dust},t} := F_{\text{remain},t} - F_{\text{LP},t}^{\text{core}} - F_{\text{BS},t}^{\text{core}} - F_{\text{TR},t}^{\text{core}} \geq 0.$$

   By construction,

$$F_{\text{LP},t}^{\text{core}} + F_{\text{BS},t}^{\text{core}} + F_{\text{TR},t}^{\text{core}} + F_{\text{dust},t} = F_{\text{remain},t}.$$

   In v1, $F_{\text{dust},t}$ is credited to LPs (equivalently, it is added to the LP residual fee share), and implementations may record $F_{\text{dust},t}$ explicitly for auditability. The final fee shares are

$$F_{\text{LP},t} := F_{\text{LP},t}^{\text{core}} + F_{\text{dust},t},$$
$$F_{\text{BS},t} := F_{\text{fill},t} + F_{\text{BS},t}^{\text{core}},$$
$$F_{\text{TR},t} := F_{\text{TR},t}^{\text{core}}.$$

Outputs and identities. Define total LP-attributed fees $F_t := F_{\text{loss},t} + F_{\text{LP},t}$. The net LP-vault credit applied at batch t is

$$\Pi_t := L_t + F_t + G_t.$$

Backstop and Treasury update by

$$B_t := B_{t-1} + F_{\text{BS},t} - G_t, \qquad T_t := T_{t-1} + F_{\text{TR},t} + (\text{other protocol flows}).$$

By construction,

$$F_{\text{tot},t} = F_{\text{loss},t} + F_{\text{LP},t} + F_{\text{BS},t} + F_{\text{TR},t},$$

and the tranche-level identities become

$$N_t^{\text{pre}} - N_{t-1} = \Pi_t = L_t + F_t + G_t,\ B_t - B_{t-1} = F_{\text{BS},t} - G_t,\ T_t - T_{t-1} = F_{\text{TR},t} + (\text{other protocol flows}).$$

Economically, loss is offset first; Backstop injects G_t only if needed to meet the drawdown floor. Remaining fees build Backstop toward coverage and then split residually across LPs/Backstop/Treasury.

4.4 Static α Cap from Entropy Budget

Under a uniform prior, the worst-case loss is bounded by $\alpha_t \ln n$. To ensure this does not exceed λE_t, we define the static cap

$$\alpha_{\text{base},t} := \frac{\lambda E_t}{\ln n}, \qquad E_t := N_{t-1}.$$

This does not depend on the choice of prior $q_{0,t}$; any incremental budget from a concentrated prior is treated as tail ΔE_t allocated to Backstop, not as a reason to shrink $\alpha_{\text{base},t}$. The Performance Layer must satisfy $\alpha_t \le \alpha_{\text{base},t}$ in every market.

4.5 Drawdown-Damped α Limit

Static caps cannot react to cumulative losses. Let P_t^{peak} be the running peak basis price up to batch t and P_t the current price; define drawdown

$$\text{DD}_t := 1 - \frac{P_t}{P_t^{\text{peak}}} \in [0, 1].$$

Given damping coefficient $k > 0$, define the next-market cap

$$\alpha_{\text{limit},t+1} := \max\{0,\ \alpha_{\text{base},t+1}(1 - k\,\text{DD}_t)\}.$$

At the end of batch t, Safety computes $E_{t+1} := N_t$, $\alpha_{\text{base},t+1}$, DD_t, and hence $\alpha_{\text{limit},t+1}$. During market $t+1$, Performance must choose depth satisfying

$$\alpha_{t+1} \le \alpha_{\text{limit},t+1}.$$

By construction $0 \le \alpha_{\text{limit},t+1} \le \alpha_{\text{base},t+1}$.

4.6 Drawdown Floor and Backstop Grants

This subsection is the canonical definition of the grant rule referenced in Section 4.3; all occurrences of G_t in capital flows use this rule.

Let N_t^{raw} be the raw post-loss NAV defined in Step 1 of the Fee Waterfall (Section 4.3), and $P_t^{\text{raw}} := N_t^{\text{raw}}/S_{t-1}$. Safety enforces the daily basis-price floor (Eq. (5)): Since $F_{\text{LP},t} \ge 0$ (dust already in $F_{\text{LP},t}$) can only increase NAV, a sufficient condition is

$$N_t^{\text{raw}} + G_t \ \ge\ (1 + p_{dd})\, P_{t-1} S_{t-1}.$$

The minimal continuous grant satisfying this is

$$G_t^{\min} := (1 + p_{dd})\, P_{t-1} S_{t-1} - N_t^{\text{raw}},$$

and on-chain

$$G_t^{\text{need}} := \max\{0, \lceil G_t^{\min} \rceil\}, \qquad G_t := G_t^{\text{need}},$$

rounded up and never negative. Safety requires $G_t^{\text{need}} \le \Delta E_t$ on admissible paths. If $G_t^{\text{need}} > \Delta E_t$, the batch reverts—the drawdown floor is an invariant, not a soft cap. If $G_t = 0$, the floor is satisfied by P&L and fees; if $G_t > 0$, Backstop injects capital to trim the tail.

By definition, $E_{\text{ent}}(q_{0,t}) = \alpha_t \ln n + \Delta E_t$, and the Safety depth cap ensures $\alpha_t \ln n \le \lambda E_t$. In the worst case, Backstop needs to cover at most the tail increment:

$$G_t \le E_{\text{ent}}(q_{0,t}) - \alpha_t \ln n = \Delta E_t.$$

Thus at most the tail ΔE_t is ever needed from Backstop to keep LP return above p_{dd}, and the admissibility rule $\Delta E_t \le B_{t-1}^{\text{eff}}$ implies Backstop is never asked to pay more than its reserved tail capacity.

Interaction with fees:

- If $F_{\text{tot},t} \ge L_t^-$, then $F_{\text{loss},t} = L_t^-$, $N_t^{\text{raw}} \ge N_{t-1}$, $r_t^{\text{raw}} \ge 0 > p_{dd}$, and $G_t = 0$.
- If $G_t > 0$, necessarily $F_{\text{tot},t} < L_t^-$, so $F_{\text{loss},t} = F_{\text{tot},t}$ and $F_{\text{pool},t} = 0$: no residual fee is distributed that day; Backstop participation is via G_t only.

With the vault identity $N_t^{\text{pre}} = N_{t-1} + L_t + F_t + G_t$ and $P_t^{\text{e}} = N_t^{\text{pre}}/S_{t-1}$, G_t is uniquely determined by (p_{dd}, α limits) and realized ($L_t, F_{\text{tot},t}$). It is strictly one-way (Backstop $\to$ LP) and keeps LP daily return above p_{dd} while B_t stays non-negative on admissible paths.

4.7 Liquidity Guards: Withdrawal Lag & LOLR

Entropy budgets and drawdown caps control solvency on the capital side; liquidity guards control solvency on the cash-flow side.

- **Fixed withdrawal lag.** As in Section 3.6, a withdrawal requested at T_{req} executes only in the first batch with $T_{\text{cycle}} \ge T_{\text{req}} + D_{\text{lag}}$, always at that batch's single price P_t^{e} (no price lock at request time). Every share present for market t absorbs that market's (L_t, F_t, G_t) before it can exit; "same market, same basis path" is preserved.
- **Treasury LOLR.** When withdrawals and losses would otherwise push depth above $\alpha_{\text{limit},t}$, Treasury may buy LP shares at batch price to raise N_t and hence E_{t+1}, restoring future depth capacity. Triggers, price rules, and limits are policy parameters (Appendix B); Treasury never pays G_t directly.

4.8 Safety Summary & Interface to Performance Layer

Summarizing:

- **Entropy budget.** $E_{\text{ent}}(q_{0,t}) = \alpha_t \ln n + \Delta E_t$, with $E_t := N_{t-1}$ and the Safety depth cap implying $\alpha_t \ln n \le \lambda E_t$ (since $\alpha_t \le \alpha_{\text{base},t} = \lambda E_t / \ln n$). The baseline term $\alpha_t \ln n$ is the uniform-prior worst-case at the chosen depth, and ΔE_t is the incremental tail induced by a concentrated prior.
- **Tranching.** LPs hold the core segment and residual fees; Backstop holds tail ΔE_t with fees $F_{\text{BS},t}$ and grants G_t; Treasury handles liquidity/ops and may buy LP shares to preserve depth, never paying G_t.
- **Depth constraint.** $\alpha_t \le \alpha_{\text{limit},t} \le \alpha_{\text{base},t} = \lambda E_t / \ln n$, with $\alpha_{\text{limit},t}$ derived from DD_{t-1} and acting as a hard cap during market t.
- **Drawdown floor.** $\dfrac{P_t^{\text{e}}}{P_{t-1}} - 1 \ge p_{dd} = -\lambda$; grants satisfy $0 \le G_t \le \Delta E_t$ and $B_t = B_{t-1} + F_{\text{BS},t} - G_t \ge 0$ on admissible priors.

- **Backstop-thin regime.** If B_{t-1} is small, admissibility forces $\Delta E_t \approx 0$: only (near-)uniform priors are allowed; LPs still face at most λE_t and the drawdown floor p_{dd} holds.
- **Liquidity guards.** Fixed D_{lag} and constrained LOLR actions prevent instantaneous depth collapse and keep one basis-price system per batch.

The Performance Layer (Section 5) treats (L_t, F_t, G_t) as observed time series and chooses depth, priors, and Fee policies *inside this safety box*: $\alpha_t \le \alpha_{\text{limit},t}$, admissible priors, fixed p_{dd} and D_{lag}, and grants as defined above. Its goals are to satisfy the long-run fee–cost condition (Section 5.4), grow LP basis price, and keep trader slippage within UX targets without weakening Safety guarantees.

5 Performance Layer – Fees, Depth, and Prior

The Performance Layer is a daily controller that chooses $(\alpha_t, q_{0,t}, \text{Fee policy})$ *inside* the Safety constraints specified in Section 4. It never modifies the Safety invariants (entropy-budget caps, drawdown floor (Eq. (5)), admissible priors, withdrawal lag, or the Fee Waterfall). Its objective is to maintain acceptable trader slippage while targeting a long-run Fee–Cost condition using realized daily metrics.

Operationally, each day follows: (i) Zero-Hour chooses $(\alpha_t, q_{0,t}, \text{Fee policy})$ under Safety caps, (ii) trading accrues $F_{\text{tot},t}$, (iii) settlement produces L_t (Eq. (4)), (iv) Safety maps to (F_t, G_t, Π_t), and (v) the batch applies Π_t and processes queues at P_t^{e}.

Sections 5.1–5.5 specify the Performance data view, Zero-Hour configuration, and daily control rules.

5.1 Daily P&L Decomposition and Monitoring Metrics

We first collect the objects that the Performance Layer treats as its "daily data view."

Daily maker P&L and pre-batch NAV. Over market t:

- L_t is the settled CLMSR maker P&L for market t (computed at settlement using settlementTick τ_t; see Section 3.4).
- $F_{\text{tot},t} \ge 0$ is the gross fee collected on trades and settlements under the Fee policy.

The Safety Layer's Fee Waterfall (Section 4.3) splits $F_{\text{tot},t}$ across loss compensation, Backstop accumulation, LP residual, Treasury, and rounding dust. From the Performance Layer's perspective, we only need the aggregated LP fee term

$$F_t := F_{\text{loss},t} + F_{\text{LP},t},$$

which is the portion of the day's fee credited to the LP Vault, and the Backstop Grant $G_t \ge 0$, which is sized by the grant rule in Section 4.6 to enforce the daily drawdown floor.

We also define the realized entropy loss

$$L_t^- := \max(-L_t, 0),$$

used below in the Fee/Loss ratio. With these definitions, LP pre-batch NAV evolves as

$$N_t^{\text{pre}} := N_{t-1} + L_t + F_t + G_t,$$

which is exactly

$$N_t^{\text{pre}} - N_{t-1} = L_t + F_t + G_t.$$

The corresponding pre-batch basis price is

$$P_t^{\text{e}} := \frac{N_t^{\text{pre}}}{S_{t-1}}.$$

Basis return. The LP's daily return is

$$R_t := \frac{P_t^{\mathrm{e}}}{P_{t-1}} - 1.$$

Safety guarantees $R_t \geq p_{dd} = -\lambda$ for all t, so the left tail of daily LP returns is truncated at a known floor by Backstop Grants.

Realized Fee/Loss ratio. To compare realized entropy cost and fee income, the Performance Layer tracks

$$\psi_{\mathrm{fee},t} := \begin{cases} \dfrac{F_t}{L_t^-}, & L_t^- > 0, \\ +\infty, & L_t^- = 0. \end{cases}$$

On loss days this is fee per unit of realized CLMSR loss; on gain days it is set to $+\infty$. Governance sets a target $\psi_{\mathrm{target}} > 1$. A moving average $\overline{\psi_{\mathrm{fee}}}$ staying above ψ_{target} is a practical translation of the long-run Fee–Cost condition $\mathbb{E}[F_t] \gtrsim \mathbb{E}[L_t^-]$.

Two-sided average slippage. For each executed trade i in market t, let v_i be the execution price and m_i the contemporaneous CLMSR midprice; define relative slippage

$$\delta_i := \frac{|v_i - m_i|}{m_i}.$$

Let Δp_t be a volume-weighted average or high percentile (e.g., 90th) of $\{\delta_i\}$ over the day. Governance specifies a UX target $\Delta p_{\mathrm{target}} > 0$. If $\Delta p_t \gg \Delta p_{\mathrm{target}}$, typical trades see excessive slippage; if $\Delta p_t \ll \Delta p_{\mathrm{target}}$, depth may be unnecessarily high for the actually traded sizes. The Performance Layer uses $(R_t, \psi_{\mathrm{fee},t}, \Delta p_t)$ as its primary signals for depth and fee adjustments.

5.2 Zero-Hour Configuration: Prior & Initial Depth

At the start of each daily market t, the CLMSR engine must be initialized at some state $q_{0,t}$ with depth α_t. Choosing both correctly reduces wasteful early-day P&L swings and makes efficient use of the entropy budget.

Exogenous prior embedding. Let p_t^0 be an exogenous estimate over ticks $b \in \mathcal{B}$. Given α_t, embed it via

$$q_{0,t,b} := \alpha_t \ln p_{t,b}^0 + \kappa_t, \quad b \in \mathcal{B},$$

where κ_t is an additive constant that leaves marginal prices unchanged. This choice affects

$$E_{\mathrm{ent}}(q_{0,t}) = \alpha_t \ln n + \Delta E_t, \quad \Delta E_t \geq 0,$$

where $\alpha_t \ln n$ is the uniform-prior baseline and ΔE_t is the tail contribution Safety assigns to the Backstop segment. Prior admissibility requires

$$\Delta E_t(q_{0,t}) \leq B_{t-1}^{\mathrm{eff}},$$

with B_{t-1}^{eff} the Backstop's effective tail capacity (Section 4). Priors that violate this bound must be flattened, combined with additional Backstop capital, or rejected.

Zero-Hour prior selection is therefore an optimization problem: approximate the settlement distribution as closely as possible while keeping ΔE_t within Backstop capacity. Estimation methods and blending rules are policy-level choices; the mechanism only requires prior admissibility.

Initial depth. Depth α_t at market open must respect Safety and UX:

- Safety gives a hard upper bound $\alpha_{\mathrm{limit},t}$ from entropy budget and drawdown (Sections 4.4–4.6).

- UX is expressed as a bound on the slippage expected for representative trade sizes selected by the off-chain UX policy. For any candidate depth α, the CLMSR curve together with those assumed sizes implies a predicted distribution of trade slippages. Let $\alpha_{\mathrm{UX},t}$ denote the depth recommended by the off-chain UX policy for market t so that representative trades are expected to see slippage near $\Delta p_{\mathrm{target}}$.

Signals v1 uses

$$\alpha_t^{\mathrm{init}} := \min\{\alpha_{\mathrm{limit},t},\, \alpha_{\mathrm{UX},t}\}.$$

If $\alpha_{\mathrm{limit},t} \geq \alpha_{\mathrm{UX},t}$, initial depth is UX-driven with Safety slack; if $\alpha_{\mathrm{limit},t} < \alpha_{\mathrm{UX},t}$, Safety dominates and slippage is temporarily higher until capital/drawdown improve. $\alpha_{\mathrm{UX},t}$ itself is not an independent governance parameter; it is derived from UX policy, current regime, and representative trade-size assumptions. After Zero-Hour, α_t is updated daily by the feedback policy in Section 5.3.

5.3 Depth Control: Squeeze / Expand Policy

Depth determines both slippage and worst-case entropy cost. Holding everything else fixed, larger α_t improves UX but amplifies potential loss; smaller α_t conserves budget but increases slippage. The Performance Layer uses a simple daily feedback controller to move α_t inside the envelope set by Safety.

Principles:

1. Depth never exceeds the Safety cap:

$$\alpha_{t+1} \leq \alpha_{\mathrm{limit},t+1}.$$

2. By default, depth slowly decreases when conditions do not justify expansion, reducing structural entropy cost over time.
3. Depth increases only when both Fee–Cost and UX indicators are favorable.

Let $\gamma \in (0,1)$ be the squeeze rate, $\eta > 0$ the expand rate, and $(\psi_{\mathrm{target}}, \Delta p_{\mathrm{target}})$ the targets.

Squeeze: Default Cost Reduction.

$$\alpha_{t+1}^{\mathrm{squeeze}} := (1-\gamma)\alpha_t.$$

This nudges depth down slowly when neither Fee–Cost nor UX signals justify expansion, conserving entropy budget. There is no separate on-chain $\alpha_{\min}$ parameter; the feedback policy uses observed slippage to decide whether expansion is justified.

Expand: Fee–Cost & UX Conditions. Expansion is considered only when

$$\psi_{\mathrm{fee},t} > \psi_{\mathrm{target}} \quad \text{and} \quad \Delta p_t > \Delta p_{\mathrm{target}}.$$

Given $\eta > 0$,

$$\alpha_{t+1}^{\mathrm{expand}} := \min\{\alpha_{\mathrm{limit},t+1},\ (1+\eta)\alpha_t\},$$

so expansion stays gradual and inside the Safety cap.

Update Rule.

$$\alpha_{t+1} = \begin{cases} \alpha_{t+1}^{\mathrm{expand}}, & \psi_{\mathrm{fee},t} > \psi_{\mathrm{target}} \text{ and } \Delta p_t > \Delta p_{\mathrm{target}}, \\ \alpha_{t+1}^{\mathrm{squeeze}}, & \text{otherwise.} \end{cases}$$

Parameters $(\gamma, \eta, \psi_{\mathrm{target}}, \Delta p_{\mathrm{target}})$ are calibrated via simulation and live data and can be tuned by governance without changing CLMSR or Safety semantics.

5.4 Fee Policy and Long-run LP Economics

Fees close the loop between CLMSR entropy cost and LP returns. From the Performance Layer's point of view there are only two moving pieces:

1. the **Fee policy**, which charges individual trades and settlements and aggregates them into a single daily gross fee $F_{\text{tot},t}$; and
2. the **Safety-fixed Fee Waterfall**, which takes $(L_t, F_{\text{tot},t}, N_{t-1}, B_{t-1})$ as input and produces the LP-facing quantities (F_t, G_t) plus Backstop/Treasury flows, according to the rules in Section 4.3.

The Performance Layer controls only the first; the second is treated as an immutable Safety mechanism.

Fee policy (how fees are collected). The Fee policy is any rule that, given the day's trading and settlement activity, produces a non-negative gross fee amount $F_{\text{tot},t}$. In practice this includes:

- maker/taker rates on range trades,
- settlement and claim fees, and
- any volume-, size-, or time-dependent adjustments (e.g., higher fees during stressed periods),

aggregated over the day.

All such choices must satisfy three constraints:

1. **Outside the cost function.** Fees are imposed *outside* $C(q)$ so that $p(q)$ keeps its probability semantics and CLMSR path-independence is preserved.
2. **Well-defined daily aggregate.** For every market t, the policy yields a unique $F_{\text{tot},t}$ that can be consumed by the Fee Waterfall in Section 4.3.
3. **Compatibility with long-run Fee–Cost inequality.** Together with depth and priors, the policy should be chosen so that the realized Fee/Loss ratio $\psi_{\text{fee},t}$ stays above a governance target $\psi_{\text{target}} > 1$ on average, making the structural condition

$$\mathbb{E}[F_t] \gtrsim \mathbb{E}[L_t^-]$$

attainable in practice.

Within these constraints the policy space is deliberately wide: Signals v1 does not fix a single fee curve. Governance instead curates a whitelist of FeePolicy implementations (contract addresses) whose published code encodes maker/taker tiers, minimum/maximum rates, and adjustment rules; the mechanism only requires that any such implementation compress its per-trade and per-settlement charges to a unique daily $F_{\text{tot},t}$.

Time-based (theta) policy example. One simple family is a Base + Theta ramp tied to trading close. Let $\tau = \max(0, T_{\text{end}} - t)$ be time remaining and W a ramp window. Then

$$r(\tau) = r_0 + \theta_{\max} \left(\frac{W - \tau}{W} \right)^{\beta} \mathbb{1}_{\tau < W},$$

and each trade pays fee = baseAmount $\cdot\, r(\tau)$ (symmetrically for buys/sells). This preserves CLMSR semantics by charging outside $C(q)$ while discouraging last-minute adverse flow as markets approach T_{end}.

Interaction with Safety, Fee Waterfall, and Grants. Once $F_{\text{tot},t}$ is known, the Safety Layer takes over. The Fee Waterfall and grant rules in Sections 4.3–4.6 map

$$(L_t,\, F_{\text{tot},t},\, N_{t-1},\, B_{t-1}) \longmapsto (F_t,\, G_t,\, F_{\text{BS},t},\, F_{\text{TR},t}),$$

with a fixed priority "loss compensation → Backstop accumulation → residual distribution."

From the Performance Layer's perspective:

- F_t is the LP-attributed fee term produced by the Fee Waterfall; it enters the daily P&L identity

  $$N_t^{\text{pre}} - N_{t-1} = L_t + F_t + G_t,$$

  and drives basis-price growth together with L_t and G_t.
- $G_t \geq 0$ is a one-way Backstop Grant fixed by Safety parameters (λ, p_{dd}, entropy budgets) and realized ($L_t, F_{\text{tot},t}$); it is not Performance-controlled.
- Backstop and Treasury flows ($F_{\text{BS},t}, F_{\text{TR},t}$) never feed back into the controller directly; they are tracked for capital-stack accounting and governance, not for daily tuning.

In other words, Performance chooses which Fee policy implementation to apply (and any operational calibration it allows) together with depth/prior policy so as to shape the joint process $(L_t, F_{\text{tot},t})$ and the derived metrics $(\psi_{\text{fee},t}, \Delta p_t)$, but it does *not* change:

- the Waterfall priority or rounding rules (Section 4.3, Appendix C),
- the grant sizing and drawdown-floor semantics (Section 4.6), or
- the Backstop coverage targets (Section 4.3) and α-limit logic (Section 4.5).

All of those belong to the Safety Layer.

Long-run LP economics under a given Fee policy. Given a fixed Safety configuration, a family of Fee policies defines a family of possible LP basis-price paths. For any such policy, the LP pre-batch NAV evolves as

$$N_t^{\text{pre}} = N_{t-1} + L_t + F_t + G_t, \qquad P_t^{\text{e}} = \frac{N_t^{\text{pre}}}{S_{t-1}},$$

and the LP's daily return is

$$R_t := \frac{P_t^{\text{e}}}{P_{t-1}} - 1 \;\geq\; p_{dd} = -\lambda.$$

The Performance Layer's objective is to choose:

- depth α_t,
- priors $q_{0,t}$, and
- the Fee policy applied to each market t (choice of whitelisted FeePolicy implementation and any parameter calibration within its published rules),

so that, under real trading behavior:

1. the long-run Fee–Cost inequality $\mathbb{E}[F_t] \gtrsim \mathbb{E}[L_t^-]$ holds,
2. empirical $\psi_{\text{fee},t}$ and slippage Δp_t stay near their governance targets, and
3. Safety invariants (entropy budgets, α-limits, drawdown floor, liquidity guards) remain strictly respected.

Signals v1 intentionally leaves the exact Fee policy functional form and calibration to governance; Section 8 describes how these parameters can be adjusted over time as more data on $(L_t, F_{\text{tot},t}, \psi_{\text{fee},t}, \Delta p_t)$ is observed.

5.5 Summary and Interface

The Performance Layer sits strictly inside the Safety envelope and operates as a daily feedback controller:

- **Contract with Safety:** depth bounded by $\alpha_t \le \alpha_{\text{limit},t}$; daily return floor $p_{dd} = -\lambda$ via grants; priors obey $\Delta E_t \le B_{t-1}^{\text{eff}}$; withdrawal lag and LOLR rules are fixed.
- **Control levers:** Zero-Hour picks $(q_{0,t}, \alpha_t)$ and Fee policy; end-of-day $(L_t, F_t, G_t, \psi_{\text{fee},t}, \Delta p_t)$ drives Squeeze/Expand and Fee policy tuning inside Safety bounds.
- **Long-run LP economics:** $N_t^{\text{pre}} - N_{t-1} = L_t + F_t + G_t$ with batch mint/burn rules; Safety trims tails and caps depth; Performance targets $\overline{\psi_{\text{fee}}} \gtrsim \psi_{\text{target}} > 1$ and $\Delta p_t \approx \Delta p_{\text{target}}$.

Future versions can introduce richer controllers—for example, multi-vault depth allocation and LP-selectable risk tranches that more finely slice which capital bears which parts of the CLMSR risk/return surface, together with LP-weighted preference aggregation and on-chain learning for priors, depth, and fee curves—so long as they respect the same Safety contract and Market-Cycle Batch accounting layer.

6 Oracle & Settlement

Signals' daily BTC range markets each have a fixed **settlement reference time** T_{set}. The BTC/USD price at this time is mapped to OutcomeSpec ticks and confirmed as the **settlement tick** τ_t, and position payouts are calculated based on this tick. The settled CLMSR maker P&L L_t is computed from $(q_{\text{start},t}, q_{\text{end},t}, \tau_t)$ as defined in Section 3.4.

In v1, settlement is divided into two layers.

- **Primary Rule**: Select one sample closest to T_{set} from the signed pull oracle feed to determine the settlement price.
- **Secondary Rule**: For markets where the primary rule fails, calculate a separate settlement price according to *pre-announced manual settlement rules* and reflect it on-chain within that day.

On-chain contracts only fix `settlementValue` and `settlementTick` after primary/secondary rules resolve. Reference CEX prices, indices, etc. for secondary settlement stay off-chain.

6.1 Signed Pull Oracle: On-chain Consumer

Signals uses a **signed pull oracle feed**. Oracle providers continuously sign and provide BTC/USD prices, and users attach this signed data to their settlement transactions and deliver it to contracts.

Oracle Packet. Settlement transaction calldata includes an oracle data packet containing the following fields.

- `feedId`: ID identifying the signed price feed.
- `price`: Signed feed price with decimals d_{feed}.
- `priceTimestamp`: UNIX timestamp when that price was observed.
- Signature and public key ID (for verifying oracle provider's signature).

Contracts parse the packet by passing `markets[marketId].feedId` to the oracle adapter function `getOracleNumericValueFromTxMsg`, then verify the signature against the registered public key.

Per-Market Oracle Configuration. Each market stores its own `feedId`, `feedDecimals`, and `tickScale`, configured at creation through the `MarketOracleConfig` argument and immutable for that market's lifetime. Settlement reads the market's stored `feedId` when asking the oracle adapter for the signed value, uses the market's `feedDecimals` to normalize the raw feed price into a fixed six-decimal `settlementValue`, and uses `tickScale` in the tick-mapping step. In v1, live markets operate on the same BTC/USD feed, but the contract surface is per-market so future asset expansion does not require a new global feed slot. The admissible sample timing constraints remain protocol-wide and governance-tunable through the slim `setRedstoneConfig(maxSampleDistance, futureTolerance)` path.

Format Constraints. After passing signature verification, contracts check the following.

- Whether `feedId` matches that market's stored feed.
- Whether the scaled `settlementValue` is bounded by `int256`. Oversized values revert, either through a Solidity arithmetic panic if scaling overflows `uint256` or through `PriceOverflow` after successful scaling.
- Whether $|\texttt{priceTimestamp} - T_{\text{set}}| \leq \Delta_{\max}$ (reject samples that are too old or early).

Nonzero pricing is an oracle-provider signing policy, not an on-chain format check: a signed `price = 0` packet would be accepted by the contract, produce `settlementValue = 0`, and then clamp to `minTick` during tick mapping. A sample satisfying the on-chain checks is normalized to an integer $x \in \mathbb{Z}$ in the fixed six-decimal `settlementValue` scale using the market's `feedDecimals`. In the deployed v1 implementation, `OracleModule` normalizes raw feed prices to this six-decimal scale regardless of OutcomeSpec d; operational BTC markets use $d = 6$, and future assets requiring another OutcomeUnit scale would need a matching implementation change in `_convertPriceToSettlementValue`. This x becomes a `settlementValue` candidate.

6.2 Outcome Domain and Tick Mapping

Outcome domain and tick mapping use the definitions from Section 2 as-is. Each market has OutcomeSpec (L, U, s, d), where L, U are OutcomeUnit-scale lower/upper bounds, s is OutcomeUnit-scale tick interval, and d is the OutcomeUnit decimal scale, distinct from the oracle config's `feedDecimals`. $U - L$ is always divisible by s, giving tick count $n = (U - L)/s$.

At the contract boundary, tick mapping is expressed in two integer steps. Each market stores a per-market `tickScale` through `MarketOracleConfig` and a `tickSpacing` through the market parameters. For BTC markets in v1, `tickScale = 1_000_000` and `tickSpacing = 200`. The scaled settlement value first maps to a raw tick value

$$\texttt{tickValue} = \left\lfloor \frac{x}{\texttt{market.tickScale}} \right\rfloor,$$

then the contract clamps that value to $[\texttt{minTick}, \texttt{maxTick} - \texttt{tickSpacing}]$ and aligns it down to the `tickSpacing` grid from `minTick`. Because primary settlement accepts non-negative signed feed prices, x is non-negative in the v1 operational regime; for non-negative values, Solidity signed integer division and the floor notation above coincide. Negative-outcome markets would require explicit handling of truncation toward zero.

At the formal OutcomeSpec level, this paper uses the corresponding bin-index representation:

$$\text{toTick}(x) = \min\left\{n - 1,\ \max\left\{0,\ \left\lfloor \tfrac{x-L}{s} \right\rfloor\right\}\right\}. \tag{6}$$

This is equivalent to the contract two-step form when $L = \texttt{minTick} \times \texttt{tickScale}$, $U = \texttt{maxTick} \times \texttt{tickScale}$, and $s = \texttt{tickSpacing} \times \texttt{tickScale}$; the aligned contract tick is $\texttt{minTick} + \text{toTick}(x) \times \texttt{tickSpacing}$. If $x < L$, clamp to the first bin; if $x \geq U$, clamp to the last bin, so edge ticks absorb prices outside the range.

On-chain, two values are ultimately stored.

- `settlementValue`: fixed six-decimal settlement price (for audit/logging).
- `settlementTick`: aligned contract tick value `minTick` + toTick(x) × `tickSpacing` stored on-chain, equivalently the value produced by `OracleTickLib.toSettlementTick`. The formal bin index τ_t = toTick(x) is the corresponding whitepaper payout object; on-chain payout logic applies the same grid mapping when using the stored aligned tick value to compute the settled maker P&L $L_t := \Delta C_t - \text{Payout}_t$ (Section 3.4).

6.3 Settlement Timeline and State Machine

Each market m has the following parameters at creation.

$$T_{\text{set}}(m), \quad \Delta_{\text{settle}}, \quad \Delta_{\text{ops}}, \quad \Delta_{\text{claim}}.$$

Default window values are governance parameters; see Appendix B. Here we define

$$T_{\text{settle,end}} := T_{\text{set}} + \Delta_{\text{settle}}, \qquad T_{\text{ops,end}} := T_{\text{settle,end}} + \Delta_{\text{ops}}.$$

The flow on the time axis is as follows.

- $t < T_{\text{set}}$: Only position trading is possible (`Trading` state).
- $T_{\text{set}} \le t < T_{\text{settle,end}}$: Anyone can call `submitSettlementSample` to submit oracle samples (`SettlementOpen` state).
- $T_{\text{settle,end}} \le t < T_{\text{ops,end}}$: No longer accept new oracle samples; stored candidate is available (`PendingOps` state). During this window, a governance-authorized operator can determine fail status with sanity checks.
- $t \ge T_{\text{settle,end}}$: If not failed and a candidate exists, a governance-authorized operator (or owner fallback) can call `finalizePrimarySettlement`. If failed, the governance owner (Safe) calls `finalizeSecondarySettle`

Market states are summarized as follows.

- `Trading`
- `SettlementOpen`
- `PendingOps`
- `FailedPendingManual`
- `FinalizedPrimary` (market confirmed by primary rule)
- `FinalizedSecondary` (market confirmed by secondary rule manual settlement)

Position claims are possible in both Finalized states, the only difference being where the settlement price came from (primary vs secondary).

State transitions are transaction-driven. `submitSettlementSample` records a candidate during `SettlementOpen`. `finalizePrimarySettlement` finalizes once `PendingOps` begins and a candidate exists. `markSettlementFailed` marks failure during `PendingOps`, and `finalizeSecondarySettlement` finalizes the manual price.

6.4 `submitSettlementSample`: Permissionless Oracle Submission

Settlement submission `submitSettlementSample(marketId)` can be called by *anyone*. It is not limited to Keepers; any address can submit a packet containing a valid oracle signature in calldata. On-chain authority control is done only through signature verification.

Time and State Constraints. `submitSettlementSample` calls must satisfy all of the following conditions.

- `block.timestamp` $\geq T_{\text{set}}$,
- `block.timestamp` $< T_{\text{settle,end}}$,
- Market state is `Trading` or `SettlementOpen`.
- Oracle payload timestamp also satisfies `priceTimestamp` $\leq$ `block.timestamp` $+ \delta_{\text{future}}$ (default $\delta_{\text{future}} = 0$) so future-dated samples are rejected.

Calls outside this range revert with `SettlementWindowClosed` type errors.

Closest Sample Selection. With a valid oracle packet, contracts convert it to (x, t_{px}) and compare with stored $(x^{(0)}, t^{(0)})$.

- If candidate is empty, store $(x^{(1)}, t^{(1)})$ as-is.
- If candidate already exists, replace with new candidate only when $|t^{(1)} - T_{\text{set}}| < |t^{(0)} - T_{\text{set}}|$.
- In case of tie, keep the more past sample (smaller t).

This way, even if update intervals are uneven around the settlement reference time, one sample closest to T_{set} is always selected.

Primary Finalization. Once `block.timestamp` $\geq T_{\text{settle,end}}$, a governance-authorized operator (or owner fallback) can call `finalizePrimarySettlement(marketId)`. This uses the stored closest-sample candidate $(x^\star, t^\star)$ to set `settlementValue`, `settlementTick`, and `settlementFinalizedAt`. After this, `submitSettlementSample` is no longer accepted and the primary settlement price does not change. A governance-authorized operator can determine off-chain whether this price passes sanity during `PendingOps`.

6.5 Claim Gating and Position Settlement

Whether primary or secondary rule, once the final settlement price is set and state is `FinalizedPrimary` or `FinalizedSecondary`, position holders claim via `claimPayout(positionId)`. Claims satisfy the following conditions.

- Market state is `FinalizedPrimary` or `FinalizedSecondary`.
- `settlementTick` is set.
- `block.timestamp` $\geq$ `settlementFinalizedAt` $+ \Delta_{\text{claim}}$.

`settlementFinalizedAt` is the transaction time that sets `FinalizedPrimary` or `FinalizedSecondary`. Claim gating is always relative to this time. With $\Delta_{\text{claim}} = 15$ minutes, claims open 15 minutes later.

Claim delay and batch accounting. Claim delay affects only when users can withdraw payouts, not the daily batch accounting. Payout liabilities are reserved (escrowed) at settlement finalization; the batch accounting for market-day t treats this reserved amount as a deducted liability when forming N_t^{pre} and the batch equity price P_t^{e}. Subsequent claim execution releases the reserved amount and does not change LP accounting (Section 3.4).

Position payouts are determined by comparing Outcome range R with final `settlementTick`, and each position ID is marked in internal state so it can only be settled once. `PositionSettled` events record position ID, owner, payout, and win/loss together so subgraph/indexer can reconstruct state.

6.6 Failure Modes and Secondary Rule

Since the primary settlement rule is not always reliable, Signals' principle is to complete manual settlement **within the same day** using *pre-announced secondary settlement rules* when it fails. Here "secondary rule" means decision rules specified in governance-published policy documents for BTC/USD data sources, sampling window, and aggregation method (VWAP/median, etc.).

Major failure modes and processing flows are as follows.

(i) Oracle Sample Absence. This is when no valid oracle samples are submitted during the settlement window, so no settlement candidate is created.

- In this case, the market enters `PendingOps`, but since the primary price is empty, primary settlement cannot be confirmed.
- A governance-authorized operator calls `markSettlementFailed(marketId)` during `PendingOps`. If no candidate exists after `PendingOps`, `markSettlementFailed` is also allowed after $T_{\text{ops,end}}$.
- Then calculate manual settlement price $x^{(2)}$ off-chain according to the secondary rule. Within the same settlement day, the governance owner (Safe) calls `finalizeSecondarySettlement(marketId, x(2))` to reflect it on-chain.

(ii) Serious Data Divergence. This is when a primary candidate price exists, but divergence from multiple CEX/indices clearly exceeds the pre-set threshold $\delta_{\max}$.

- A governance-authorized operator calls `markSettlementFailed(marketId)` within the `PendingOps` period (about 5 minutes) to mark as Failed. This call is not allowed for anyone, only for the governance authority (owner/multisig).
- The subsequent procedure is the same as (i). According to the secondary rule, calculate new settlement price $x^{(2)}$ off-chain and reflect on-chain with `finalizeSecondarySettlement` by the governance owner (Safe) within that day.

On-chain Reflection of Secondary Settlement. `finalizeSecondarySettlement(marketId, x(2))` can only be called once for Failed state markets and performs the following.

- Verify market state is `FailedPendingManual` and reject markets where manual settlement is already done.
- Verify $x^{(2)}$ as an OutcomeUnit integer and apply `toTick` to calculate `settlementTickSecondary`.
- Set `settlementValue` to $x^{(2)}$, `settlementTick` to that tick, and record `settlementFinalizedAt` as `block.timestamp`.
- Transition state to `FinalizedSecondary`.

The authorized operator SHOULD submit the manual settlement within the same settlement day (e.g., $[T_{\text{set}}, T_{\text{set}} + 24\text{h})$) so markets do not remain stranded in Failed state. After this, claims work the same as primary settlement; only `settlementTick` comes from the secondary rule price.

(iii) Keeper Failure. Since oracle sample submission is permissionless, even if a specific Keeper fails, another party can call `submitSettlementSample` with the same oracle packet. If no one sends `submitSettlementSample` within the settle window, it results in (i), and then manual settlement is done with the secondary rule.

Manual Settlement Within the Day. Fail means "primary price not trusted; settle manually the same day." Data sources/windows/aggregation for the secondary rule live in governance policy documents; this spec fixes only on-chain invariants (time gates, state transitions, single execution). `settlementTick` is set once and drives CLMSR P&L and position payouts.

7 System Architecture

This section provides a high-level on-chain architecture overview: the object model, unit conventions, and the market/vault state machines. Concrete storage layouts, data-structure choices, and library-level implementation details are collected in Appendix D.

7.1 Implementation Scope & Object Model

This section covers the on-chain objects that implement the semantic objects from Sections 2–6:

- **Market.** OutcomeSpec, depth α, fee policy, timing, per-market oracle configuration (`feedId`, `feedDecimals`, `tickScale`), and settlement fields are stored in a per-market on-chain record.
- **CLMSR pricing state.** The maker maintains strictly positive weights $\{w_b\}$ (equivalently, the softmax state $\mathbf{q}$) and supports (i) multiplicative range updates corresponding to position changes and (ii) range-sum queries needed for pricing via Z and Z_R.
- **Exposure ledger (settlement accounting).** The system tracks outstanding payout liabilities per tick $Q_{t,b}$, supporting range-add updates on open/increase/decrease/close and a point query at the realized settlement tick to compute $\text{Payout}_t = Q_{t,\tau_t}$.
- **LP Vault.** Maker capital is held in an ERC–4626-style vault whose share accounting follows the daily Market-Cycle Batch semantics (Section 3) and stores $(N, S, P, P^{\text{peak}}, \text{DD})$.
- **Risk state.** Safety/Performance parameters (depth caps, drawdown tracking, and coverage targets) are stored in configuration; the v1 on-chain enforcement surface is market creation, while vault batch parameters are applied during batch finalization.
- **Positions.** Each trader position is represented as an ERC–721 token carrying $(marketId, R, x)$ and a settled flag.
- **Oracle/settlement.** Settlement consumes signed oracle data, stores `settlementValue` and `settlementTick` exactly once per market, and gates claims according to the settlement state machine (Section 6).

Each bullet above is a responsibility boundary: mechanism math defines semantics (CLMSR updates, batch accounting), contracts enforce validation and lifecycle, and off-chain infra (frontends, oracle operators, indexers) remains out of scope. The concrete reference implementation choices are described in Appendix D.

7.2 Global Unit System & Fixed-Point Arithmetic

Units. External token transfers use the settlement token's native decimals (e.g., USDC 6 decimals). Internal CLMSR math uses WAD (10^{18}). Outcome/settlement values use the integer scale 10^d from `OutcomeSpec` (L, U, s, d) (Section 6). In v1, `OracleModule` produces a fixed six-decimal `settlementValue`, so the operational OutcomeUnit scale is $d = 6$; the formal 10^d form remains the spec object for future asset expansion, but matching d to another settlement scale would require an implementation update. Tick mapping then divides the scaled settlement value by the market's `tickScale`, clamps the raw tick to the market bounds, and aligns it to OutcomeSpec spacing.

Conversions. Token $\leftrightarrow$ WAD conversions occur once on entry to and exit from CLMSR math. The rule is *one conversion, one rounding* per direction to avoid rounding arbitrage.

Rounding policy. Trade costs/fees debit users with rounding *up*; trade proceeds credit users with rounding *down*. Deposit/withdrawal, fee split, and grant rounding follow the vault rules in Section 3.8; rounding cannot make the protocol pay more than the WAD-level economics.

Arithmetic. All internal calculations use 10^{18} fixed-point arithmetic with explicit overflow/underflow guards. Full numeric/rounding rules are fixed in Appendix C; invariant and testing surfaces live in Appendix D.

7.3 Market and Vault State Machines

Core and modules. The on-chain system is organized into a core contract that owns storage and exposes external entrypoints, with modules responsible for market lifecycle (create/settle/fail/manual settle), trading (open/increase/decrease/close/claim), vault batching, risk enforcement, and oracle payload validation. Upgrade and wiring details are described in Appendix D.

Market state machine. Mirrors Section 6: states progress through `Trading`, `SettlementOpen`, then `PendingOps` or `FailedPendingManual`, and finally `FinalizedPrimary` or `FinalizedSecondary`. Each transition is owned by a specific function, and settlement is one-time per market and immutable once set.

Batch execution. The daily LP batch is executed by the governance authority. `processDailyBatch` may be called once per market-day after that market has reached a finalized state; calls revert if the market is not yet `FinalizedPrimary`/`FinalizedSecondary` or if the batch for that day already ran. Claim and Δ_{claim} delays gate position claims, not batch execution.

In v1, batch execution finalizes $(N_t^{\text{pre}}, P_t^{\text{e}})$ and records request assignments; individual LP deposit/withdraw requests are settled via per-request claims after finalization, without changing P_t^{e}.

Upgrade and role boundaries. Mechanism semantics (CLMSR math, rounding invariants) are treated as fixed; upgrades focus on configuration and module wiring under governance controls. All state-changing entrypoints are guarded by roles and scoped pausability.

Vault batch. Section 3's N_t, S_t, P_t update runs in one daily batch: apply the net LP-vault credit Π_t (derived from L_t, F_t, G_t by the Safety Layer), compute pre/post NAV, process the withdraw queue then deposit queue, and record (N_t, S_t, P_t). Invariants: $P_t = N_t/S_t$ within rounding; shares that rode the same markets see identical basis changes; rounding dust stays internal except the explicit deposit refunds in Section 3.8(b1).

Risk gate. The v1 `RiskModule` enforcement surface is `gateCreateMarket`: it checks α limits and prior admissibility before a market is listed. Drawdown affects the $\alpha_{\text{limit},t}$ calculation; trade entrypoints and LP-vault functions are outside the `RiskModule` gate surface in v1.

8 Governance Model & Roles

This section specifies (i) what is immutable mechanism semantics, (ii) what is configurable on-chain, (iii) what remains off-chain operational policy, and (iv) the roles and authorities of key actors. All core mechanism semantics for Signals v1 are defined in Sections 2–7. Parameter catalog details are in Appendix B.

8.1 Governance Surface and Variability

From a governance perspective, Signals v1 has three distinct layers:

1. Mechanism layer (immutable semantics, changed only by versioned upgrades).
2. On-chain configuration layer (governance parameters).
3. Off-chain operational policy layer.

(1) Mechanism layer (immutable semantics) The following elements define the core product and can change only via explicit protocol upgrades:

- **CLMSR semantics:** cost function/prices (Eqs. (1)–(2)), range trades, and settlement ledger (Section 2).
- **Batch/Vault semantics:** daily batch price (Eq. (3)), mint/burn invariants, and withdrawal lag (Section 3).

- **Safety semantics:** entropy budgets, α-caps, drawdown floor (Eq. (5)), Fee Waterfall, and grant rule (Section 4).
- **Oracle/settlement semantics:** state machine, timing windows, and claim gating (Section 6).

These items are treated as part of the protocol's contractual semantics for daily BTC range markets.

(2) On-chain configuration layer (governance parameters) The following values are configurable parameters that appear in the Safety, Performance, and oracle layers. They are stored in dedicated configuration contracts and can be changed by governance subject to the processes in Section 9.

- **Safety parameters** (Section 4):
  risk-budget parameter λ (with drawdown floor $p_{\mathrm{dd}} := -\lambda$ derived from it), drawdown damping coefficient k for $\alpha_{\mathrm{limit},t}$, withdrawal lag D_{lag}, and Backstop coverage targets.
- **Performance parameters** (Section 5):
  squeeze rate γ, expand rate η, Fee/Loss target ψ_{target}, slippage target $\Delta p_{\mathrm{target}}$, the whitelist of FeePolicy implementations (contract addresses), and fee-split weights for LP/Backstop/Treasury.
- **Market/product parameters** (Sections 2, 3, 6):
  OutcomeSpec templates (L, U, s, d) for BTC markets and market calendar (e.g., UTC time for T_{set}, trading close).
- **Oracle/settlement parameters** (Section 6):
  settlement window Δ_{settle}, PendingOps window Δ_{ops}, claim delay Δ_{claim}, allowed timestamp deviation $\Delta_{\max}$, and divergence threshold $\delta_{\max}$.

Per-market `feedId`, `feedDecimals`, and `tickScale` are configured at market creation via `MarketOracleConfig` and are immutable for that market's lifetime. They are part of the market record rather than the governance-tunable configuration layer. Oracle signer identities and provider key management remain external trust-boundary policy, while the on-chain protocol enforces signed payload verification and the governance-tunable sample timing constraints.

Core contracts read the governance-tunable values above from configuration contracts; they do not allow arbitrary direct writes. Only governance-authorized keys can update configuration, and such updates are controlled by timelocks and access control described in Section 9.

(3) Off-chain operational policy layer Some rules are procedural rather than on-chain invariants. They are documented in governance-published policy documents and must remain consistent with the invariant surface above:

- data sources and algorithms for secondary settlement (CEX set, indices, sampling windows, aggregation method such as VWAP/median/TWAP),
- models and procedures for prior estimation at Zero-Hour (combining options markets, historical distributions, internal forecasts) used to build $q_{0,t}$,
- off-chain criteria for `markSettlementFailed` decisions (e.g., divergence beyond $\delta_{\max}$, oracle outages),
- Treasury capital policy for LP share purchases (LOLR) and Backstop recapitalization,
- UX policies for displaying settlement windows, withdrawal delay, drawdown caps, and market states in frontends.

These policies can evolve more frequently than on-chain configuration, but they may not override or circumvent contract-level invariants.

8.2 Roles and Authorities

Signals v1 involves several categories of participants. This subsection summarizes, for each one, the main exposures, sources of return, and authorities.

(1) LP Vault LPs LPs supply capital to the LP Vault (Section 3) and hold ERC-4626-style shares representing proportional claims on NAV N_t.

- **Exposure.**
  LPs are exposed to the daily pre-batch P&L
  $$\Pi_t = L_t + F_t + G_t$$
  where L_t is CLMSR maker P&L, F_t is LP-attributed fees, and $G_t \geq 0$ is the Backstop Grant (Sections 3.4, 5.1). Under the Safety Layer's depth limits, drawdown cap, and prior-admissibility rule, daily basis-price returns are truncated at p_{dd} or better (Section 4.5).
- **Compensation.**
  LPs participate in basis-price growth $P_t = N_t/S_t$. Fees that remain after loss compensation and Backstop accumulation accrue to N_t via F_t; if the long-run fee–cost condition holds (Section 5.4), P_t grows structurally over time.
- **Rights.**
  LPs may submit deposit/withdrawal *requests* and have them settled after delay D_{lag} at the batch price P_t^{e} via per-request claims (Sections 3.5, 3.6), and may transfer LP shares freely or create secondary markets on external venues (Section 3.7).

(2) Backstop Vault providers Backstop providers supply capital to the Backstop Vault (Section 4.2), which holds mezzanine tail risk.

- **Exposure.**
  Backstop pays Grants $G_t > 0$ on days when raw LP returns would violate the drawdown cap p_{dd}. Under the prior-admissibility rule $\Delta E_t \leq B_{t-1}^{\mathrm{eff}} \leq B_{t-1}$ (Section 4.1), the worst-case grant per day is bounded by ΔE_t, so the Backstop Vault remains solvent on admissible paths (Sections 4.1, 4.6).
- **Compensation.**
  Backstop receives a fee share $F_{\mathrm{BS},t}$ from the daily Waterfall (Section 4.3). Economically, this is a portion of LP fee income carved out to build insurance reserves. When tails do not materialize, this fee share accumulates; when tails hit, reserves are drawn down via G_t.
- **Rights.**
  Backstop providers have claims on Backstop NAV B_t and may, depending on governance design, participate in parameter decisions that primarily affect tail risk (e.g., Backstop fee share, coverage targets).

(3) Treasury Treasury is a protocol-level capital account that plays two roles (Section 4.2):

- **Lender of Last Resort (LOLR).**
  In stress scenarios where withdrawals and losses cause N_t to fall and depth would otherwise exceed $\alpha_{\mathrm{limit},t}$, Treasury can buy LP shares at the batch price, increasing N_t, restoring $E_{t+1} := N_t$ and future depth capacity (Sections 4.5, 9).
- **Operating capital.**
  Treasury funds audits, upgrades, incident response, and may recapitalize Backstop if its NAV falls below target. These flows are governed decisions, not automatic reactions.

Treasury never pays G_t directly; all Backstop Grants are internal transfers from Backstop to LP Vault. In v1, entities controlling Treasury typically also control configuration contracts and upgrade proxies, subject to the constraints in Section 9.

(4) Oracle providers, keepers, and authorized operators Oracle and settlement responsibilities are split as follows (Section 6):

- **Oracle providers** maintain signed BTC/USD feeds that satisfy freshness and quality constraints. They are responsible for data integrity and availability within the allowed timestamp window $\Delta_{\max}$.
- **Keepers** (any addresses) can call `submitSettlementSample` during the settlement window. They are not trusted for content, only for liveness; all critical checks (signature, feedId, timing, normalization) are enforced on-chain.
- **Operations accounts** are governance-controlled multisigs with limited powers:
  - call `markSettlementFailed` during PendingOps for markets whose primary price fails sanity checks,
  - call `finalizeSecondarySettlement` once with the secondary-settlement price computed according to published rules (Section 6).

(5) Frontends and integrators Frontends and external protocols are not part of the core mechanism but materially affect user risk perception:

- Frontends must accurately display settlement times, windows, the withdrawal delay D_{lag}, drawdown caps, and market states.
- Integrations that use LP shares or position NFTs as collateral must clearly indicate any additional leverage, liquidation logic, or risk transformation beyond what Signals itself defines.

These actors do not have protocol-level authority, but their design choices influence how effectively users can interpret the on-chain semantics.

9 Change Process & Decentralization

Governance must be able to adjust parameters and, when necessary, take emergency actions without breaking the mechanism semantics or surprising LPs and traders. This section describes the normal change process, emergency tools, and the relationship to secondary settlement. Section 9.4 then sketches a decentralization roadmap, and Section 9.5 summarizes trust assumptions and the threat model.

9.1 Normal parameter changes

In v1 it is useful to distinguish between:

- **controller outputs**, which are expected to move every market-day (for example the next-day depth α_t chosen inside Safety caps, the Zero-Hour prior $q_{0,t}$, and the Fee policy selected for each market from a whitelisted family); and
- **configuration parameters**, which define the Safety envelope and these controllers (for example λ, k, D_{lag}, Backstop coverage targets, depth and slippage targets, oracle windows, and the whitelist of allowed FeePolicy implementations and product templates).

Daily tuning of controller outputs inside the Safety envelope — choosing $(\alpha_t, q_{0,t}, \text{Fee policy})$ for each market t with $\alpha_t \le \alpha_{\text{limit},t}$ and admissible priors — is treated as normal operation and does not itself go through the governance proposal process.

By contrast, changing configuration parameters (adjusting λ or k, modifying Backstop coverage targets or fee-split weights, altering oracle sample-timing windows, or changing the set or ranges of allowed FeePolicy implementations and OutcomeSpec templates) follows a standard governance flow:

Routine changes to Safety, Performance, oracle sample-timing, and product configuration parameters follow a standard flow:

1. **Proposal.**
   Governance participants publish a proposal specifying:
   - the parameters and before/after values,
   - the rationale and expected impact on Safety, Performance, and UX,
   - any dependencies (e.g., if one change assumes another has been executed).
2. **Off-chain review.** A minimum review period allows LPs, Backstop providers, and other stakeholders to comment via forums or other governance channels.
3. **On-chain timelock.** Approved proposals are queued in configuration contracts with a timelock. Safety-critical parameters (e.g., λ, D_{lag}, Δ_{settle}) may have longer timelocks and higher approval thresholds than purely UX-related ones.
4. **Execution.** After the timelock, authorized addresses call configuration setters (e.g., `setRiskConfig`, `setFeeWaterfallConfig`, `setRedstoneConfig`).
   Events record the change and the proposal ID for auditability.

9.2 Emergency measures and circuit breakers

In exceptional conditions (oracle failures, discovered vulnerabilities, Backstop depletion, governance compromise risk), an emergency multisig may have limited authority to take fast, scoped actions without going through the full proposal cycle. Typical measures include:

- temporarily pausing new market creation or new trading for specific markets,
- temporarily pausing *risk-increasing* actions (e.g., new market creation, new trading, or new request submission), while keeping *risk-decreasing / obligation-settling* actions (e.g., per-position payout claims and matured vault request claims) available so users can exit and the protocol can honor finalized obligations,
- forcing depth control into "squeeze only" mode (disabling expansion) for new cycles,
- disabling leverage in connected products where applicable.

Emergency actions must be strictly scoped (which functions/markets are affected), be time-bounded (with explicit sunset or requirement for formal ratification), and be reflected in on-chain state and off-chain communications so users can detect them.

9.3 Secondary settlement process

Section 6 defines secondary settlement for markets where the primary rule fails. Governance-authorized operators apply it as follows:

1. During PendingOps, an authorized operator may call `markSettlementFailed` for a market whose primary candidate price is missing or obviously invalid (e.g., divergence beyond $\delta_{\max}$ from reference data).
2. Off-chain, the authorized operator computes a secondary settlement price $x^{(2)}$ and timestamp $t^{(2)}$ according to a published secondary rule (data sources, sampling window, aggregation method).
3. Within the same settlement day, the authorized operator calls `finalizeSecondarySettlement(marketId, x)` once to set `settlementValue` and `settlementTick` from that secondary price and transition the market to `FinalizedSecondary`.

Claim gating and single-settlement semantics remain unchanged: positions can be claimed only after the claim delay Δ_{claim}, and each position can be settled exactly once.

9.4 Progressive Decentralization Roadmap

Signals v1 starts with governed configuration and scoped operational roles (oracle failure handling, emergency pausing). Over time, the protocol aims to minimize privileged authority by locking core semantics and limiting change to transparent parameter updates under timelocks.

9.5 Trust Assumptions and Threat Model

Key risks include market regime shifts, oracle failures/manipulation, smart contract vulnerabilities, and coordinated liquidity outflows. Mitigations are defined by the Safety envelope (entropy budgets, α-limits, drawdown floor, Backstop grants), the settlement state machine, and governance controls (multisig + timelocks + emergency scopes). This section is a summary; detailed operational policies are maintained off-chain and must remain consistent with on-chain invariants.

10 Conclusion

Signals v1 specifies daily BTC range markets implemented via a CLMSR pricing engine (Eqs. (1)–(2)), a batched LP vault (Eq. (3)), and an explicit Safety/Performance split.

CLMSR provides unified liquidity across a discretized outcome axis with bounded worst-case loss, while the Market-Cycle Batch maps daily maker outcomes into a single equity price for LPs. Safety enforces solvency invariants (entropy budgets, drawdown floor (Eq. (5)), grants, and liquidity guards), and Performance operates as a controller inside that envelope.

Next steps include: (i) empirical calibration of priors, depth, and fee policies under live data, (ii) extending the product surface beyond a single asset/tenor while preserving the same Safety contract, and (iii) hardening oracle reliability and implementation invariants through audits and monitoring.

A Notation & State Variable Table

This appendix summarizes the core symbols used throughout the paper. Each symbol is defined where it first appears; this table is a cross-reference only.

A.1 CLMSR and Outcome Space

Symbol	Meaning
(L, U, s, d)	OutcomeSpec: lower/upper bounds, tick spacing, decimals (Sections 2, 6)
$\mathcal{B}, n$	Tick set $\mathcal{B} = \{0, \dots, n-1\}$, tick count $n = (U-L)/s$
q, q_i	CLMSR pricing state (real-valued log-weights; may be negative for non-uniform priors)
w_b	Pricing weight $w_b := e^{q_b/\alpha} > 0$ (on-chain implementations store w_b and sums)
$Q_{t,b}$	Outstanding exposure ledger (integer): total settlement-token amount owed if $\tau_t = b$ (Section 3.4)
$C(q)$	CLMSR cost function $C(q) = \alpha \ln Z(q)$
$Z(q)$	Partition sum $Z(q) = \sum_b e^{q_b/\alpha} = \sum_b w_b$
$p_b(q)$	Tick prices/probabilities $p_b(q) = \partial C/\partial q_b = w_b/Z$
$R \subseteq \mathcal{B}$	Contiguous range of bins; on-chain represented by boundary ticks $(\texttt{lowerTick}, \texttt{upperTick})$ meaning $\texttt{lowerTick} \le b < \texttt{upperTick}$

A.2 Daily P&L and Fees

Symbol	Meaning
ΔC_t	Gross trading cashflow (path-independent): $\Delta C_t := C(q_{\text{end},t}) - C(q_{\text{start},t})$
Payout_t	Settlement payout: $\text{Payout}_t := Q_{t,\tau_t}$ (outstanding exposure at settlement tick)
L_t	Settled maker P&L: $L_t := \Delta C_t - \text{Payout}_t$ (Section 3.4)
L_t^-	Realized entropy loss $L_t^- := \max(-L_t, 0)$
$F_{\text{tot},t}$	Gross trading/settlement fees generated in market t
$F_{\text{loss},t}$	Portion of $F_{\text{tot},t}$ used immediately to offset CLMSR loss
$F_{\text{pool},t}$	Remaining fee pool after loss compensation: $F_{\text{pool},t} := F_{\text{tot},t} - F_{\text{loss},t}$
$F_{\text{LP},t}$	Residual fee share allocated to the LP Vault (after Backstop target filling)
$F_{\text{BS},t}$	Fee share allocated to the Backstop Vault (coverage fill + residual share)
$F_{\text{TR},t}$	Fee share allocated to Treasury
F_t	LP-attributed fee term: $F_t := F_{\text{loss},t} + F_{\text{LP},t}$
$\psi_{\text{fee},t}$	Realized Fee/Loss ratio: $\psi_{\text{fee},t} := F_t / L_t^-$ if $L_t^- > 0$, else $+\infty$
ψ_{target}	Governance target for long-run Fee/Loss ratio
$F_{\text{dust},t}$	Rounding residual in the residual-fee split; credited to LPs and included in $F_{\text{LP},t}$

A.3 Capital Accounts and Batch States

Symbol	Meaning
N_t	LP Vault NAV at end of batch t
S_t	LP Vault share count at end of batch t
P_t	LP Vault basis price $P_t := N_t / S_t$ at end of batch t
P_t^{peak}	Peak basis price to date, $P_t^{\text{peak}} := \max_{\tau \le t} P_\tau$
DD_t	Drawdown $\text{DD}_t := 1 - P_t / P_t^{\text{peak}}$
E_t	LP capital at start of market t ($E_t := N_{t-1}$)
N_t^{pre}	Pre-batch NAV after daily P&L: $N_t^{\text{pre}} := N_{t-1} + L_t + F_t + G_t$
P_t^{e}	Batch equity price applied to existing shares: $P_t^{\text{e}} := N_t^{\text{pre}} / S_{t-1}$
B_t	Backstop Vault NAV at end of batch t
T_t	Treasury NAV at end of batch t
G_t	Backstop Grant (Backstop → LP Vault) for drawdown-cap enforcement
Π_t	LP Vault daily P&L (pre-batch): $\Pi_t := N_t^{\text{pre}} - N_{t-1} = L_t + F_t + G_t$

Raw NAV/price variables used only in Safety/Waterfall definitions:

Symbol	Meaning
N_t^{raw}	Raw NAV after CLMSR P&L and loss-offset fees: $N_t^{\text{raw}} := N_{t-1} + L_t + F_{\text{loss},t}$
P_t^{raw}	Raw basis price before grant: $P_t^{\text{raw}} := N_t^{\text{raw}} / S_{t-1}$

These appear only in Section 4.6 to size G_t.

A.4 Safety and Performance Parameters

Symbol	Meaning
λ	Dimensionless risk-budget parameter; under a uniform prior the static depth cap is $\alpha_{\text{base},t} = \lambda E_t / \ln n$ so that $\alpha_t \ln n \le \lambda E_t$; the drawdown cap is $p_{dd} := -\lambda$ (Sections 4.1, 4.4)
α_t	Depth used in daily market t
$\alpha_{\text{base},t}$	Static depth cap from entropy budget/capital (Section 4.4)
$\alpha_{\text{limit},t}$	Drawdown-damped depth cap, $\alpha_{\text{limit},t} \le \alpha_{\text{base},t}$
$E_{\text{ent}}(q_{0,t})$	Entropy loss budget for market t from opening state $q_{0,t}$ (Section 4)
ΔE_t	Incremental entropy-tail budget: $\Delta E_t := E_{\text{ent}}(q_{0,t}) - \alpha_t \ln n$; allocated to Backstop (Sections 4.1, 4.2)
p_{dd}	Daily drawdown cap for LP basis-price return, defined as $p_{dd} := -\lambda$
D_{lag}	Fixed withdrawal lag (request → batch execution)
ρ_{BS}	Backstop coverage target ratio (Backstop NAV vs LP NAV), Appendix B
Δp_t	Two-sided average slippage indicator for trades in market t (Section 5.1)
Δp_{target}	Governance slippage target

The full parameter catalog (fee-schedule parameters, oracle windows, market templates) is in Appendix B.

B Parameter Catalog

This appendix lists only governance-tunable parameters. Formal definitions and formulas are in Sections 4–5 (Safety/Performance) and Section 6 (Oracle); implementation-level use lives in Section 7.

Safety Parameters

λ Effective single-market worst-case loss ratio under a uniform prior; the static cap $\alpha_{\text{base},t} := \lambda E_t / \ln n$ ensures $\alpha_t \ln n \le \lambda E_t$, and $p_{dd} := -\lambda$ is derived from this.

p_{dd} Daily drawdown cap for LP basis price (defined as $p_{dd} = -\lambda$).

k Drawdown decay coefficient used in $\alpha_{\text{limit},t}$.

D_{lag} Fixed withdrawal delay.

Backstop coverage target Target ratio of Backstop NAV to LP Vault NAV.

Performance Parameters

γ Depth squeeze rate.

η Depth expand rate.

ψ_{target} Long-run Fee/Loss ratio target.

Δp_{target} Slippage target for typical trades (see Section 5.1).

Fee policy whitelist Addresses of whitelisted FeePolicy implementations that map trades/settlements to $F_{\text{tot},t}$.

Fee Waterfall weights Residual split ratios for LP/Backstop/Treasury after loss compensation and Backstop coverage fill (e.g., parameters $(\phi_{\text{LP}}, \phi_{\text{BS}})$ with $\phi_{\text{TR}} := 1 - \phi_{\text{LP}} - \phi_{\text{BS}}$).

Market/Product Parameters

(L, U, s, d) OutcomeSpec (domain bounds, tick spacing, decimals).

$n_{\max}$ Maximum bin count (v1: $n_{\max} = 256$). OutcomeSpec templates must satisfy $n = (U-L)/s \le n_{\max}$ for gas/DoS safety.

T_{set} Settlement reference time (trading closes just before T_{set}; see Section 6).

Oracle/Settlement Parameters

Δ_{settle} Settlement window length.

Δ_{ops} PendingOps window length.

Δ_{claim} Claim delay after Finalized.

$\Delta_{\max}$ Allowed price timestamp deviation.

$\delta_{\max}$ Divergence threshold for `markSettlementFailed`.

C Numerics & Rounding Rules

External settlement currency is USDC 6 decimals, internal calculations are processed in WAD (10^{18}). For a single transaction, only **one conversion, one rounding** of "external→internal→cost/price calculation→external" is allowed to prevent rounding arbitrage. Fees are imposed and distributed outside the cost function to preserve the normalization/additivity semantics that interpret p as "probability". Trade/LP/fee/grant rounding and dust rules are summarized in Section 3.8; this appendix and Section 7.2 are the canonical specifications.

C.1 Rounding and Dust Handling in the Vault

Price representation. In this appendix, $\widetilde{P}_t$ denotes the on-chain fixed-point representation of the batch equity price P_t^{e} used for asset↔share conversions in batch t (i.e., the single conversion price derived from $P_t^{\text{e}} = N_t^{\text{pre}}/S_{t-1}$ under the vault's WAD arithmetic and rounding rules).

All vault accounting is in integer settlement-token units; rounding and dust ownership are fixed as follows so that the continuous invariant $N'/S' = N/S$ from Section 3.3 is respected up to at most one base unit of error.

(a) Trade rounding. CLMSR costs/proceeds are computed in WAD, then converted to the 6-dec token with *debits rounded up, credits rounded down*. Per-trade dust is at most one base unit and stays on the maker side, flowing into L_t or F_t (see Section 7.2).

(b1) Deposits (asset→share). Deposits are submitted as *requests* and may escrow assets immediately. The conversion rate is determined only once the assigned batch is finalized at price $\widetilde{P}_t$.

For a deposit request of amount A assigned to batch t:

$$S_{\text{mint}} := \left\lfloor \frac{A}{\widetilde{P}_t} \right\rfloor, \qquad A_{\text{used}} := S_{\text{mint}} \cdot \widetilde{P}_t, \qquad A_{\text{refund}} := A - A_{\text{used}} \ge 0.$$

The depositor becomes entitled to $(S_{\text{mint}}, A_{\text{refund}})$ once batch t is finalized. The refund is delivered via the per-request claim flow (or recorded as claimable refund); escrowed but not-yet-claimed amounts are segregated from the vault's accounting NAV and cannot be used for market-making.

(b2) Withdrawals (share→asset). Burn x shares and pay

$$A_{\text{wd}} := \left\lfloor x \cdot \widetilde{P}_t \right\rfloor .$$

Rounding dust $x\widetilde{P}_t - A_{\text{wd}} \in [0, 1)$ stays in the vault and accrues to remaining LPs via a slightly higher basis price. Credits are always truncated, so users never receive more than the WAD-level value.

(c) Fee waterfall splits (including explicit dust tracking). Let $F_t^{\text{remain}} \geq 0$ be the integer fee pool remaining after loss compensation and any Backstop-coverage fill in the Fee Waterfall (Steps 1–3 of Section 4.3). Given governance weights $(\phi_{\text{LP}}, \phi_{\text{BS}}, \phi_{\text{TR}})$ with $\phi_{\text{LP}} + \phi_{\text{BS}} + \phi_{\text{TR}} = 1$, compute core shares and dust:

$$F_{\text{LP},t}^{\text{core}} := \left\lfloor F_t^{\text{remain}} \cdot \phi_{\text{LP}} \right\rfloor , \quad F_{\text{BS},t}^{\text{core}} := \left\lfloor F_t^{\text{remain}} \cdot \phi_{\text{BS}} \right\rfloor , \quad F_{\text{TR},t}^{\text{core}} := \left\lfloor F_t^{\text{remain}} \cdot \phi_{\text{TR}} \right\rfloor ,$$

$$F_{\text{dust},t} := F_t^{\text{remain}} - F_{\text{LP},t}^{\text{core}} - F_{\text{BS},t}^{\text{core}} - F_{\text{TR},t}^{\text{core}} \geq 0.$$

Set $F_{\text{LP},t} := F_{\text{LP},t}^{\text{core}} + F_{\text{dust},t}$, $F_{\text{BS},t} := F_{\text{fill},t} + F_{\text{BS},t}^{\text{core}}$, $F_{\text{TR},t} := F_{\text{TR},t}^{\text{core}}$. By construction $F_{\text{LP},t}^{\text{core}} + F_{\text{BS},t}^{\text{core}} + F_{\text{TR},t}^{\text{core}} + F_{\text{dust},t} = F_t^{\text{remain}}$, and v1 credits $F_{\text{dust},t}$ to LPs while optionally recording it explicitly for auditability.

The LP fee term entering daily P&L is

$$F_t := F_{\text{loss},t} + F_{\text{LP},t},$$

consistent with Sections 3.4 and 5.1. Backstop and Treasury fee shares update as $B_t - B_{t-1} = F_{\text{BS},t} - G_t$ and $T_t - T_{t-1} = F_{\text{TR},t} + (\text{other flows})$; rounding never causes payouts to exceed WAD-level economics.

(d) Backstop Grants G_t (tail cut, one-way). From the continuous recommendation

$$G_{\min,t} = (1 + p_{dd}) P_{t-1} S_{t-1} - N_t^{\text{raw}}, \qquad N_t^{\text{raw}} := N_{t-1} + L_t + F_{\text{loss},t},$$

on-chain uses

$$G_t := \max\{0, \lceil G_{\min,t} \rceil\},$$

decreasing Backstop and increasing the Vault by the same integer G_t. This conservatively enforces the drawdown cap with at most one base unit of extra cost. All Backstop upside and recovery is modeled via its fee waterfall share $F_{\text{BS},t}$.

D Implementation Blueprint & Testing

D.1 CLMSR Engine: Tree, Cost, and Complexity

State representation. CLMSR weights $w_b = e^{q_b/\alpha}$ are stored in a lazy range-multiplication segment tree. Leaves store w_b (WAD), internal nodes cache subtree sums, and pending factors accumulate lazy updates. The root sum is $Z(q)$.

Range updates/queries. Increasing a range position by x on $[l, h]$ applies factor $e^{x/\alpha}$ to that interval via `applyFactor` in $O(\log n)$. Range sums `sumRange` return Z_R. Both methods honor pending factors and keep `cachedRootSum` synchronized.

Safe exponent and chunking. Exponent input $z = (\Delta q/\alpha)$ is constrained to $z \leq z_{\max}$. If a trade would exceed this, the library computes a safe chunk size deterministically from inputs and applies the trade in multiple chunks, each within domain, summing costs so that path-independence holds at WAD precision.

Invariants/tests. Tree invariants: $w_b > 0$; increasing a position on positive-mass ranges yields $Z' > Z$; root sum matches composed factors. Property tests include round-trip checks (open then close the same range/size approximately restores distribution) and partition normalization.

D.2 Oracle Adapter and Settlement

OracleModule. Stores per-market oracle feed/config, verifies signatures and staleness, checks $|t - T_{\text{set}}| \leq \Delta_{\max}$, converts price to outcome units, and exposes a single interface to obtain a candidate settlement price.

`submitSettlementSample`. Permissionless; validates the payload and updates the closest-sample candidate during `SettlementOpen`. It does not finalize settlement. Finalization is done by a governance-authorized operator via `finalizePrimarySettlement`, or by `markSettlementFailed` followed by `finalizeSecondarySet` In both finalized paths, `settlementTick` is set exactly once.

Snapshots/indexing. Post-settlement, `requestSettlementChunks` emits bounded chunk events to let off-chain indexers process positions without per-position on-chain events; claims use on-chain view math so on/off-chain payouts match.

D.3 Upgradeability, Config, and Security Guards

Upgrade pattern. Only `SignalsCore` and `SignalsPosition` are UUPS proxies. Modules are replaceable logic targets called via `delegatecall`. Storage layout is centralized in `SignalsCoreStorage` and `SignalsPositionStorage`.

Config layer. `SignalsCoreStorage` holds governance parameters (fees, α caps, λ, $\Delta_{\max}$, D_{lag}, exposure limits). Modules read these values when enforcing the invariants from Sections 4–6.

Guards. External entrypoints use `nonReentrant`/`whenNotPaused` and role checks. Oracle/settlement functions validate feed IDs, timestamps, and signature keys. Token interactions are constrained to whitelisted assets. Pausing and owner hooks are scoped to lifecycle/config changes, not to math semantics.

D.4 Invariants, Testing, and Spec-as-Code

Invariants (summary). CLMSR: non-negative weights, path-independent costs, loss bound via $\alpha \ln n$, partition normalization. Market/Vault: one settlement per market; $N_t^{\text{pre}} - N_{t-1} = L_t + F_t + G_t$ (pre-batch); shares riding the same markets see the same basis changes; withdrawal lag and drawdown caps are enforced; Backstop balance follows $B_{t+1} = B_t + F_{\text{BS},t} - G_t$. Oracle: single `settlementTick` with time gates and $\Delta_{\max}$.

Testing surface. Unit tests for validation/state machines; parity tests for CLMSR math vs. reference engine; property tests for tree consistency and round-trips; batch/NAV tests for vault accounting; oracle/settlement edge-case tests. v0 parity tests cover SDK semantics from earlier internal versions where applicable.

Spec-as-code. The whitepaper (Sections 2–7) and code-level invariant docs/tests are kept in sync; changing mechanism libraries or their invariants is treated as a protocol version change, not an implementation detail.

References

[1] R. Hanson. Logarithmic market scoring rules for modular combinatorial information aggregation. *Journal of Prediction Markets*, 1(1):3–15, 2007.

[2] J. Abernethy, Y. Chen, and J. Wortman Vaughan. An optimization-based framework for automated market-making. In *Proceedings of the 14th ACM Conference on Electronic Commerce*, pages 297–314, 2013.

[3] J. Wolfers and E. Zitzewitz. Prediction markets. *Journal of Economic Perspectives*, 18(2):107–126, 2004.

[4] J. Moody et al. EIP-4626: Tokenized Vault Standard. Ethereum Improvement Proposal 4626, 2022.

[5] RedStone Team. RedStone Oracles Documentation. https://docs.redstone.finance/, accessed 2025.

[6] Chainway Labs. Citrea: Bitcoin-native ZK Rollup. https://citrea.xyz/, accessed 2025.

[7] Signals Research Team. CLMSR Mechanism Whitepaper. Technical report, Signals Protocol, 2025.

Disclaimer

This document is provided for informational and technical purposes only and does not constitute financial, investment, legal, accounting, or tax advice.

It is not an offer or solicitation to buy or sell any token, security, derivative, or other instrument.

The protocol is experimental and may result in partial or total loss due to market risk, oracle failures, implementation bugs, governance/operational risks, and adversarial behavior.

Regulatory treatment varies by jurisdiction and may change over time.

In any inconsistency between this document and deployed smart contracts, the behavior of deployed code prevails.