Contents

Overview of LS-LMSR Market Design for RetroPick as ShadowPool: Off-chain &On-Chain Prediction and Liquidity Provision

Asyam Jayanegara
jayanegara.asyam@gmail.com

Delphi runs an on-chain automated market maker (AMM) for prediction markets based on the Logarithmic Market Scoring Rule (LMSR). These markets trade claims of the form "Model *i* wins the competition" where each claim pays one token if and only if the corresponding model wins, and zero otherwise. LMSR guarantees prices are continuous, always available, and behave like probabilities. The market is backed by a community-supplied vault that earns trading fees while taking on a bounded worst-case loss, calibrated via the LMSR liquidity parameter.

In this report we give a technical overview of Delphi's market microstructure. We formalize the LMSR cost-function market maker; show how it induces prices, trading, and a bounded loss guarantee; describe the vault and fee mechanism that allow external liquidity providers to back the AMM; and discuss risk budgeting and parameter selection. Finally, we explain how this microstructure will compose with reproducible, verifiable settlement for machine-learning competitions using refereed delegation frameworks such as Verde (5).

Corresponding author(s): jayanegara.asyam@gmail.com

1. Introduction

Prediction markets trade contingent claims on future events, i.e. trades encode the traders' belief that a future event will occur. When well-designed and sufficiently liquid, they can aggregate information and produce prices that behave like calibrated probabilities (6). A central design choice for prediction markets is the mechanism used to quote and clear trades:

- *Order-book markets* match buyers and sellers through a central limit order book or continuous double auction; risk is largely borne by *traders* and any designated market makers.
- *Cost-function market makers* (CFMMs) maintain a convex cost function C over outstanding share quantities and quote prices as its gradient (2). The venue (or its liquidity providers) takes on a *bounded* worst-case loss in exchange for always-available liquidity.

Unlike other high-volume prediction market platforms today, Delphi adopts the second approach using the Logarithmic Market Scoring Rule (LS-LMSR) introduced by Hanson (1). LS-LMSR is a particular cost function that has been extensively studied and is well-suited to the market claims in Delphi because it:

1. Provides continuous prices that sum to one and can be interpreted as probabilities,
2. Has a closed-form bound on the market maker's worst-case loss,
3. Is simple to implement on-chain, requiring only a handful of pure arithmetic operations per trade.

Currently, Delphi runs LS-LMSR markets over the outcomes of machine-learning competitions:

- Each outcome corresponds to a model $i \in \{1, \ldots, n\}$ winning the evaluation.
- A share of outcome i pays 1 token if model i wins, and 0 otherwise.
- Exactly one model wins, so the payoff vector is a standard basis vector.

Importantly, Delphi does not rely on human resolution, committee votes, or oracles to resolve outcomes and settle markets. Instead settlement will be (once feature complete) tied to a fully reproducible, verifiable, and transparent evaluation pipeline—algorithmic judges determine the winner and are integrated with Verde (5) to ensure results can be disputed if called into question.

This report focuses on the market microstructure; we sketch settlement only briefly. We assume the reader is comfortable with basic probability, optimization, and DeFi-style AMMs.

2. Setting & Background

2.1. Outcome space and securities

Let $\Omega = \{1, \ldots, n\}$ denote a finite set of mutually exclusive and exhaustive outcomes. In RetroPick model-winner market:

$$\Omega = \{1, \ldots, n\} = \{\text{“Model 1 wins”}, \ldots, \text{“Model } n \text{ wins”}\}, \text{and}$$

exactly one outcome $\omega \in \Omega$ is realized at settlement.

We consider a vector of Arrow-Debreu securities $X(\omega) \in \mathbb{R}^n$ with components

$$X_i(\omega) = \begin{cases} 1, & \omega = i, \\ 0, & \omega \neq i. \end{cases}$$

A share of security i pays 1 if outcome i occurs and 0 otherwise. A trader's position is a vector $\theta \in \mathbb{R}^n$, where θ_i is the number of shares of outcome i they hold.

2.2. Cost-function market makers

A cost-function market maker maintains a differentiable, convex function $C : \mathbb{R}^n \to \mathbb{R}$ mapping vectors $q \in \mathbb{R}^n$ of outstanding shares held by all traders to an aggregate market value. Intuitively, $C(q)$ is the cumulative amount of base tokens that have been paid into the market maker to create the current state q. That said, as will be discussed throughout this report, the power of these cost-functions is not usually in the value of $C(q)$ directly; it is in the rich market-wide information encoded by higher-order properties of C at any given market state q.

Given a current state q:

- The *instantaneous price* vector is
$$p(q) = \nabla C(q) \in \mathbb{R}^n.$$
- A trader who wants to execute some trade vector $\Delta \in \mathbb{R}^n$ and, hence, change the global share vector from q to $q + \Delta$ pays
$$\text{TradeCost}(q, \Delta) = C(q + \Delta) - C(q).$$

Thus the market maker is always willing to "sell" $\Delta_i > 0$ shares of outcome i at a total cost equal to the increase in C, and similarly will "buy back" shares when $\Delta_i < 0$.

Cost-function market makers are mathematically equivalent to market scoring rules (MSRs) derived from proper scoring rules and utility-based market makers that maintain a fixed level of expected utility over outcome wealth (2). In Delphi we use LS-LMSR, which corresponds to the log scoring rule and to a negative exponential-utility market maker. For the remainder of this report we focus discussion on facets of LS-LMSR relevant to understanding Delphi, but the literature on cost-function markets is vast and worth exploring[1].

3. The Logarithmic Market Scoring Rule (LS-LMSR)

3.1. Definition

LS-LMSR is defined by the cost function
$$C(q) = b \log \left(\sum_{i=1}^{n} \exp\left(\frac{q_i}{b} \right) \right), \quad (1)$$
where $b > 0$ is a *liquidity* (or *depth*) parameter. All quantities are denominated in the base token (e.g. $AI). The gradient of C is
$$p_i(q) = \frac{\partial C}{\partial q_i}(q) = \frac{\exp(q_i/b)}{\sum_{j=1}^{n} \exp(q_j/b)}, \qquad i = 1, \ldots, n, \quad (2)$$
which is just a logit map. Therefore:
$$p_i(q) \in (0, 1) \quad \text{and} \quad \sum_{i=1}^{n} p_i(q) = 1.$$

Hence the price vector $p(q)$ can thus be interpreted as a *probability distribution* over outcomes.

[1] The foundations laid by Frongillo & Waggoner (3) is a good starting point for exploring MSRs and cost-function markets.

3.2. Trading and execution prices

Consider a trade that increments holdings of outcome k by $\Delta \in \mathbb{R}$ (positive for a buy, negative for a sell), so the state moves from q to $q' = q + \Delta e_k$ where e_k is the k-th unit vector. The total cost (from the trader's perspective) is

$$\Delta C = C(q + \Delta e_k) - C(q). \tag{3}$$

The *average execution price* per share is

$$\bar{p}_k(q, \Delta) = \frac{\Delta C}{\Delta} \quad \text{for } \Delta \neq 0.$$

By convexity of C we have

$$\Delta C \geq \nabla C(q) \cdot (\Delta e_k) = p_k(q)\,\Delta, \quad \text{when } \Delta > 0,$$

and

$$\Delta C \leq \nabla C(q') \cdot (\Delta e_k) = p_k(q')\,\Delta, \quad \text{when } \Delta < 0.$$

Thus for a buy ($\Delta > 0$), $\bar{p}_k(q, \Delta)$ lies between the initial price $p_k(q)$ and the final price $p_k(q')$. Furthermore, larger $|\Delta|$ induces more price movement ("slippage"), governed by the curvature (i.e. the Hessian) of C.

3.3. Worst-case loss bound

Suppose the market maker initially has state $q = 0$ and cash $B_0 = -C(0)$ (so that its net wealth is normalized to zero). After an arbitrary sequence of trades, the state is q and the cumulated cash held by the market maker is $B = -C(0) + C(q)$.

When outcome $\omega \in \Omega$ is realized, the market maker must pay q_ω tokens to holders of outcome ω. Its terminal wealth is thus

$$W(\omega) = B - q_\omega = C(q) - C(0) - q_\omega.$$

The worst-case loss is

$$L_{\max} = \sup_{q} \max_{\omega} [q_\omega - C(q) + C(0)] = \sup_{q} \max_{i} [q_i - C(q)] + C(0).$$

For LS-LMSR with payouts in $[0, 1]$ and $C(0) = b \log n$, one can show (1; 2) that

$$L_{\max} \leq b \log n. \tag{4}$$

Intuitively, as traders push one outcome's probability toward 1 (and others toward 0), they must pay in more and more to move the log partition function, limiting the severity of a subsequent upset.

In Delphi we treat $L_{\max}$ as a *risk budget* allocated to the market and choose b accordingly:

$$b = \frac{L_{\max}}{\log n}. \tag{5}$$

3.4. Interpretation as a proper scoring rule

LS-LMSR is the market-scoring-rule version of the (logarithmic) proper scoring rule. In the single-trader case, if a forecaster reports probability vector p and the realized outcome is ω, then their log score is proportional to $\log p_\omega$. Truthful reporting of their belief distribution maximizes the expected log score.

In a market scoring rule traders sequentially update the current quote p to a new quote p' and pay the difference in score relative to a reference prediction. Hanson (1) shows that, in the cost-function formulation, the logarithmic scoring rule corresponds exactly to the cost function in equation 1. Chen and Pennock (2) further characterize LS-LMSR as an exponential-utility market maker.

4. RetroPick LS-LMSR Market Specification

Delphi instantiates this general LS-LMSR framework in specialized model-competition markets. In this section we explain how trades, fees, and settlement works.

4.1. Model-winner outcomes

Let $\mathsf{M} = \{1, \ldots, n\}$ be the index set of submitted models in a competition. The outcome space is $\Omega = \mathsf{M}$, and the payoff of a share of model i is $X_i(\omega) = \mathbb{1}[\omega = i]$, with the constraint that only one outcome wins.

The evaluation pipeline (§8) computes a deterministic function

$$w : \mathsf{M} \to \Omega$$

that maps the set of possible models to a unique winner, e.g. the model with the highest score under a fixed metric on a fixed dataset (with pre-committed tie-breaking).

4.2. State variables

At any block height t, the on-chain state of a single market includes:

- Share vector $q(t) = (q_1(t), \ldots, q_n(t)) \in \mathbb{R}^n$;
- LS-LMSR liquidity parameter $b > 0$ (constant over the life of a market);
- Fee parameter $\tau \in [0, 1)$ (constant or piecewise constant);
- Vault balance $V(t)$ (see §5);
- Revenue pool balance $R(t)$ (accumulated net fees).

We suppress explicit time indices when unambiguous.

On-chain, these quantities are stored as integers in a fixed-point format (e.g. 10^{-18} token precision). The functional form of C and p are implemented using exponentials and logarithms built into Solidity's PRBMath library, with appropriate thresholding on trades to ensure numerical and market stability.

4.3. Fee mechanism

Delphi charges a proportional fee $\tau \in [0, 1)$ on the notional size of each trade. For a state transition $q \to q' = q + \Delta e_k$, recall that

$$\Delta C = C(q') - C(q)$$

and define the notional $N = |\Delta C|$. We are ready to derive actualized costs per trade and fee revenues.

Buy-side trade (i.e. $\Delta C > 0$). The trader pays

$$\text{CashOut} = (1 + \tau)\, \Delta C$$

tokens, i.e. cost to buy shares and the trading fees. The AMM's internal cash account increases by ΔC, and the revenue pool increases by $\Delta R = \tau \Delta C$.

Sell-side trade (i.e. $\Delta C < 0$). The trader receives

$$\text{CashIn} = (1 - \tau)\, |\Delta C|$$

tokens, i.e. return on shares and the trading fees. The AMM releases $|\Delta C|$ from its cash account, and the revenue pool receives $\Delta R = \tau |\Delta C|$.

Effect on bounded loss. The classical LS-LMSR bound (equation 4) assumes no fees. With fees, the worst-case *net loss* to the AMM is strictly less, because every trade contributes a non-negative amount to $R(t)$, which remains available to cover payouts at settlement.

In practice, we treat the theoretical bound $L_{\max} = b \log n$ as a conservative risk budget and layer additional safety margins on top of it (§5.3).

5. LS-LMSR Vault: Liquidity Provision and P&L Sharing

The LS-LMSR Vault is a pooled capital account that backs the AMM and earns trading fees. During testnet the Vault is entirely provisioned by the market maker (i.e. Gensyn), but in the future it will allow external participants to act as passive market makers.

5.1. Vault mechanics

Let $V(t)$ be the total value of the vault (in \$AI) at time t, and let $S(t)$ be the total supply of a vault share token.

Deposits. When a user deposits an amount $d > 0$ at time t, they receive

$$s_{\text{mint}} = \begin{cases} d, & \text{if } S(t) = 0 \text{ (first depositor)}, \\ d \cdot \dfrac{S(t)}{V(t)}, & \text{otherwise,} \end{cases}$$

new vault share tokens, preserving proportional ownership. The share supply increases to $S(t^+) = S(t) + s_{\text{mint}}$ and the vault balance to $V(t^+) = V(t) + d$.

Withdrawals. When a user redeems $s > 0$ vault shares, they receive

$$d_{\text{redeem}} = s \cdot \frac{V(t)}{S(t)}$$

tokens, and the share supply decreases to $S(t^+) = S(t) - s$, with $V(t^+) = V(t) - d_{\text{redeem}}$.

Trading P&L. For a single market, the vault experiences:

- Fee inflows ΔR on every trade (§4.3),
- A terminal payoff at settlement equal to *minus* the AMM's net liability to winning traders.

Let F denote total fee income and L_{real} the realized loss (if any) of the LS-LMSR for that market. Then the contribution of that market to the vault's value is

$$\Delta V = F - L_{\text{real}}.$$

Across many markets, these contributions aggregate over time.

5.2. Break-even turnover and risk budget

Let L be the worst-case loss budget assigned to a given market (e.g. $L = b \log n$). Suppose the total absolute trade notional (ignoring sign) over the life of the market is:

$$V = \sum_{\text{trades } k} |\Delta C_k|.$$

Fee income is approximately $F \approx \tau V$.

A simple heuristic for break-even turnover V^* is

$$V^* \approx \frac{L}{\tau}, \tag{6}$$

Therefore, if $V \gg L/\tau$, fee income alone can cover the worst-case loss before accounting for any favorable trading P&L.

Illustrative example. Suppose there are $n = 10$ models, a risk budget $L = 40{,}000$ tokens for the LS-LMSR, fee of $\tau = 1\%$, and realized volume $V = 8{,}000{,}000$ tokens. Then

$$F \approx 0.01 \cdot 8{,}000{,}000 = 80{,}000,$$

and even if L_{real} hits the full budget $L = 40{,}000$, the vault is ahead by 40,000 tokens before costs. Note that this is not a guarantee of profit; it is a sizing heuristic for b and τ.

5.3. Coverage and risk management

We define a *coverage ratio* for the vault relative to the total risk budget allocated across active markets:

$$\kappa(t) = \frac{V(t)}{\sum_{m \in \mathcal{M}_{\text{active}}} L_m},$$

where L_m is the loss budget for market m. We target $\kappa(t)$ above a threshold, e.g. $\kappa_{\min} > 1$. If $\kappa(t)$ approaches the threshold, protocol-level controls can automatically:

- Reduce b (and thus L_m) on new or even existing markets;
- Impose caps on order sizes to limit rapid changes in q;
- Temporarily pause new market creation or deposits.

Vault deposits and withdrawals are also subject to windows around settlement times to avoid last-minute liquidity flight. These parameters are design decisions that can be tuned based on empirical usage.

6. Risk Calibration and the Liquidity Parameter

6.1. From worst-case loss to depth

Given a desired per-market loss budget L and number of outcomes n, we select

$$b = \frac{L}{\log n}.$$

The choice of b implies a specific trade-off:

- Near uniform prices ($p \approx 1/n$), what impact do small trades have on price.

- Moving an outcome's implied probability from p to p' requires total cost roughly $b \log \frac{p'}{p}$ (holding others fixed), so larger b increases the monetary cost of large mov

6.2. Pathwise considerations

The bound $L_{\max} \leq b \log n$ is path-independent: it holds regardless of the sequence of trades. In practice, realized loss L_{real} is often much smaller because:

- Traders partially "self-insure" when they buy and later sell as beliefs change.
- Two-sided flow (some traders buying, others selling) tends to keep q closer to balanced states where p_i remain near uniform.

However, for protocol safety we do not rely on this; risk controls are based on the worst-case bound.

7. Comparison with Order-Book Prediction Markets

Many large prediction venues (e.g. Polymarket and Kalshi) implement central limit order books or continuous double auctions for trading event contracts, especially in regulated settings. These designs primarily act as *matching engines*: traders (or external market makers) post collateral and take on outcome risk, while the venue itself can externalize most inventory risk (4).

From the perspective of our design space:

- **Order books (matching designs).** The venue creates and destroys outcome shares only when there is a counterparty; properly margined, the venue itself need not hold a risky inventory. Hence liquidity is endogenous and can dry up in thin markets, leading to wide spreads and stale prices.
- **Cost-function AMMs (LS-LMSR).** The venue (or its liquidity providers) runs an always-on market maker willing to buy or sell any bundle at prices given by $\nabla C(q)$. Inventory risk is borne by the market maker, but is *provably bounded* and can be calibrated via b.

For RetroPick specific use case—thin yet information-rich competitions—LS-LMSR has several advantages:

- Continuous liquidity even when very few traders are active;
- Coherent probabilities across all models where prices sum to 1 by construction;
- A clean, parametrically bounded risk profile for Market Makers.

8. Settlement and Reproducible Evaluation

The payout of RetroPick LS-LMSR market depends on the outcome of an ML competition. To make settlement reproducible and verifiable, we will rely on two components:

1. A fully specified evaluation program E (model loading, preprocessing, metrics, tie-breaking);
2. Verde's (5) refereed-delegation protocol and reproducible operator library which ensures independent evaluators obtain bitwise-identical results.

Initially, Delphi will utilize transparent public reporting of evaluations as we build out and test features modularly. Once the Verde protocol's integration is feature complete, settlement will be fully reproducible and verifiable by machine.

8.1. Evaluation as a reproducible program

Let E denote a deterministic program that, given:
- A model index i,
- A dataset D,
- Fixed hyperparameters and seeds,

returns a scalar score $\sigma_i \in \mathbb{R}$ (or a vector of scores). The winner function is

$$w = \arg\max_{i \in \{1,\dots,n\}} \sigma_i,$$

with deterministic tie-breaking if necessary.

In practice, both E and D are committed on-chain before trading opens. The commitment ensures that the evaluation definition cannot be changed ex post.

8.2. Verifiable execution via refereed delegation

Verde (5) adapts the cryptographic notion of *refereed delegation* to machine learning programs. At a high level:

- Multiple compute providers can independently run the evaluation program E and commit to check- points and outputs via Merkle-tree-based hashes.
- If providers disagree on the output, a referee runs an interactive dispute resolution protocol that recursively narrows down the first diverging checkpoint and then the first diverging operation in the underlying computational graph.
- At the lowest level, the referee re-executes a single operator to determine which party is honest.

To make this viable, Verde relies on a library of *reproducible operators* (RepOps) that enforce a deterministic execution order for floating-point operations across hardware setups. This avoids the usual non-determinism introduced by parallel floating-point arithmetic on GPUs.

For Delphi, the key property is:

> *Any honest verifier (e.g. a node, user, or governance process) can re-run the evaluation of the winning model using the same program E and RepOps, and obtain the same bitwise output. If any evaluator deviates, a refereed-delegation protocol can economically punish them. Hence markets can be transparently settled by machines rather than opaque resolution mechanisms.*

Thus, the mapping from market state (q, b, τ) to settlement outcome ω is:

$$\omega = w\, E(1), \ldots, E(n) \ ,$$

where each $E(i)$ is reproducible, and disputes about E can be resolved via a dispute resolution game.

8.3. Composition with the LS-LMSR AMM

From the perspective of the LS-LMSR AMM, settlement requires only the index ω of the winning outcome. Once ω is determined:

- The protocol computes each trader's net position in shares of outcome ω,
- Pays out 1 token per share,
- Burns all outstanding shares (or marks the market as resolved),
- Realizes the AMM's P&L relative to the vault.

Ultimately the result is that the market participants will effectively bid on verifiable outputs of de- terministic ML evaluations, rather than on a vaguely defined event subject to discretionary human resolution.

9. Conclusion and Extensions

We described RetroPick LS-LMSR-based prediction market design, which is backed by a community vault and paired with verifiable ML settlement. The main design choices were:

- Using LS-LMSR to obtain continuous, probabilistic prices and a bounded worst-case loss;
- Funding the market maker via an on-chain vault that shares fee income and risk among stakers;
- Calibrating the liquidity parameter b and fee τ to satisfy a risk budget and break-even turnover target;
- Integrating with Verde's refereed-delegation framework making settlement reproducible and verifiable.

From an engineering perspective, LS-LMSR provides a compact and mathematically tractable core: a single convex function and its gradient. Around this, Delphi layers the practical concerns of risk management, liquidity provision, and reproducible ML evaluation, resulting in a prediction market system that is truly decentralized, transparent, and computationally robust.

References

[1] R. Hanson. Logarithmic Market Scoring Rules for Modular Combinatorial Information Aggregation. *Journal of Prediction Markets*, 1(1):3–15, 2007. Available at https://mason.gmu.edu/~rhanson/mktscore.pdf.

[2] Y. Chen and D. M. Pennock. A Utility Framework for Bounded-Loss Market Makers. arXiv:1206.5252, 2012.

[3] R. Frongillo and B. Waggoner. An Axiomatic Study of Scoring Rule Markets. *Innovations in Theoretical Computer Science Conference (ITCS)*, 2018.

[4] N. Rahman, J. Al-Chami, and J. Clark. SoK: Market Microstructure for Decentralized Prediction Markets (DePMs). arXiv:2510.15612, 2025.

[5] A. Arun, A. St. Arnaud, A. Titov, B. Wilcox, V. Kolobaric, M. Brinkmann, O. Ersoy, B. Fielding, and J. Bonneau. Verde: Verification via Refereed Delegation for Machine Learning Programs. arXiv:2502.19405, 2025.

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