ISOMETRIC PROTOCOL

On-Chain Architecture

Technical Overview

Version 1.1 — March 2026

An AMM for Prediction Markets on Solana

Table of Contents

1. Protocol Overview & Architecture

Isometric is an automated market maker (AMM) for prediction markets on Solana. It merges Meteora-style discrete liquidity bins with continuous outcome speculation, enabling range-based predictions with graded payouts instead of binary yes/no outcomes.

Core Innovation

- Range-Weighted LMSR: Per-bin liquidity parameters proportional to LP capital deposits
- Gaussian Kernel Payouts: Continuous payout distribution (no binary cliff edges)
- Single-Sided LP: LPs deposit only USDC and select outcome ranges to underwrite
- Sigmoid Bonding Curves: Local supply/demand pricing within each range bucket
- Dutch Auction Liquidations: Descending-price auctions for leveraged position liquidation

Technology Stack

- Blockchain: Solana (devnet initially, mainnet-beta for production)
- Framework: Anchor
- Token Standard: SPL Token / Token-2022 for position representations
- Oracles: Pyth Network (primary), Switchboard (fallback)
- Settlement Asset: USDC

2. Program Decomposition

The protocol is split into four independent but composable programs. Each is deployed separately and communicates via Cross-Program Invocation (CPI).

Program	Responsibility	Key Accounts
iso_core	LMSR pricing engine, market creation, position entry/exit, sigmoid bonding curves	Market, Bin, Position, MarketConfig
iso_vault	USDC custody, LP deposits/withdrawals, range selection, fee accrual	Vault, LPPosition, BinLiquidity
iso_settlement	Gaussian kernel payout computation, market resolution, TWAP finalization	Settlement, PayoutRecord
iso_liquidation	VaR monitoring, Dutch auction creation/execution, margin enforcement	Auction, LiquidationConfig

CPI Dependency Graph

- iso_core → iso_vault (reads bin liquidity to compute b_k)
- iso_core → Pyth/Switchboard (reads price feeds for spot reference)
- iso_settlement → iso_core (reads positions for payout computation)
- iso_settlement → iso_vault (transfers USDC payouts from vaults)
- iso_liquidation → iso_core (marks positions as liquidated)
- iso_liquidation → iso_vault (transfers margin to auction winners)

3. Account Structures (PDAs)

All state is stored in Program Derived Addresses (PDAs). Seeds are specified for each account to ensure deterministic derivation.

3.1 Market

Seeds: [b"market", market_id.to_le_bytes()]

Field	Type	Description
authority	Pubkey	Market creator (admin)
market_id	u64	Unique market identifier
title	String (64)	Human-readable market title
description	String (256)	Market description
oracle_feed	Pubkey	Pyth/Switchboard feed account
outcome_range_min	u64	Minimum outcome value (scaled by 10^6)
outcome_range_max	u64	Maximum outcome value (scaled by 10^6)
bin_count	u16	Number of discrete bins (e.g., 8, 16, 32)
sigma	u64	Gaussian payout spread parameter (scaled 10^6)
b_base	u64	Base LMSR liquidity parameter (scaled 10^6)
resolution_timestamp	i64	Unix timestamp for market resolution
twap_window_slots	u32	Number of slots for TWAP settlement (~900 = 30 min)
status	MarketStatus	Enum: Active, ResolutionPending, Settled, Cancelled
resolved_outcome	Option<u64>	Final TWAP outcome after settlement (scaled 10^6)
total_volume	u64	Cumulative trading volume in USDC
created_at	i64	Creation timestamp
bump	u8	PDA bump seed

3.2 Bin

Seeds: [b"bin", market_id.to_le_bytes(), bin_index.to_le_bytes()]

Field	Type	Description
market	Pubkey	Parent market account
bin_index	u16	Index within market (0 to bin_count-1)
range_low	u64	Lower bound of this bin (scaled 10^6)
range_high	u64	Upper bound of this bin (scaled 10^6)

liquidity	u64	Total LP capital deposited in this bin (USDC, 10^6)
shares_outstanding	u64	Total shares sold in this bin
b_k	u64	Computed: b_base * (liquidity / avg_liquidity)
bump	u8	PDA bump

3.3 Position (Trader)

Seeds: [b"position", market_id.to_le_bytes(), owner.as_ref(), position_nonce.to_le_bytes()]

Field	Type	Description
owner	Pubkey	Position holder wallet
market	Pubkey	Market account
range_low	u64	Lower bound of prediction range (scaled 10^6)
range_high	u64	Upper bound of prediction range (scaled 10^6)
shares	u64	Number of shares held (scaled 10^6)
entry_cost	u64	USDC paid at entry (scaled 10^6)
leverage	u16	Leverage multiplier (100 = 1x, 200 = 2x, etc.)
margin_deposited	u64	USDC margin for leveraged positions
entry_timestamp	i64	When position was opened
status	PositionStatus	Enum: Active, Liquidatable, Settled, Closed
nonce	u64	Unique nonce per owner per market
bump	u8	PDA bump

3.4 LPPosition

Seeds: [b"lp_position", market_id.to_le_bytes(), owner.as_ref(), lp_nonce.to_le_bytes()]

Field	Type	Description
owner	Pubkey	LP wallet
market	Pubkey	Market account
deposit_amount	u64	USDC deposited (scaled 10^6)
range_low	u64	Lower bound of LP range
range_high	u64	Upper bound of LP range
fees_earned	u64	Accumulated trading fees
entry_timestamp	i64	When LP position was created
status	LPStatus	Enum: Active, PendingWithdrawal, Withdrawn
nonce	u64	Unique nonce
bump	u8	PDA bump

3.5 Vault

Seeds: [b"vault", market_id.to_le_bytes()]

Field	Type	Description
market	Pubkey	Associated market
usdc_mint	Pubkey	USDC mint address
token_account	Pubkey	PDA-owned USDC token account
total_deposits	u64	Sum of all LP deposits
total_reserved	u64	Funds reserved for open positions/payouts
insurance_fund_balance	u64	Insurance fund allocation for this market
bump	u8	PDA bump

3.6 Auction (Liquidation)

Seeds: [b"auction", position.as_ref()]

Field	Type	Description
position	Pubkey	Position being liquidated
start_price	u64	Starting auction price (105% of est. value)
end_price	u64	Minimum auction price (floor)
decay_rate_bps	u16	Price decay in basis points per slot
start_slot	u64	Slot when auction began
max_auction_slots	u32	Maximum slots before expiry (~2700 = 18 min)
status	AuctionStatus	Enum: Active, Claimed, Expired
claimer	Option<Pubkey>	Address that claimed the auction
claimed_price	Option<u64>	Final price paid by claimer
bump	u8	PDA bump

4. Core Program: Range-Weighted LMSR Engine

4.1 Liquidity Parameter Computation

For each bin k with LP capital L_k:

b_k = b_base × (L_k / L_avg)

Where L_avg is the average liquidity across all active bins. Bins with deeper liquidity produce tighter spreads.

4.2 Cost Function

The cost to buy Δq shares in bin k follows the LMSR formula:

Cost = b_k × ln(e^((q_k + Δq) / b_k) + ∑ e^(q_j / b_k)) − b_k × ln(∑ e^(q_j / b_k))

All math is performed in fixed-point u128 with 10^{12} scaling to maintain precision.

4.3 Sigmoid Entry Pricing

Within each bin, a local sigmoid bonding curve adjusts entry price:

entry_price = P_min + (P_max − P_min) / (1 + e^(−k(q − q_mid)))

4.4 Range Overlap Computation

When a trader's range spans multiple bins, shares are distributed proportionally:

overlap_k = max(0, min(range_high, bin_high) − max(range_low, bin_low)) / (range_high − range_low)

Total cost sums across all overlapping bins.

4.5 open_position Logic

- Validate market is Active
- Validate range_low < range_high and within market bounds
- Compute bin overlaps for the requested range
- For each overlapping bin: compute LMSR cost with updated b_k
- Apply sigmoid entry pricing modifier
- Sum total cost across all bins
- Assert total_cost ≤ max_cost (slippage protection)
- Transfer USDC from user to vault
- Create Position PDA
- Emit PositionOpened event

5. Vault Program: LP Capital Management

5.1 deposit_liquidity

- LP specifies market, range [low, high], and USDC amount
- Compute which bins overlap with LP’s range
- Distribute deposit proportionally across overlapping bins
- Update each bin’s liquidity and recompute b_k
- Transfer USDC from LP to vault token account
- Create LPPosition PDA
- Emit LiquidityDeposited event

5.2 withdraw_liquidity

- Assert market is Settled or LP requests early withdrawal
- Compute current value of LP’s share across bins
- Compute impermanent loss (IL)
- If IL > 5% of deposit_amount: insurance_payout = IL − (0.05 × deposit_amount)
- Transfer deposit + fees − IL + insurance_payout to LP
- Update bin liquidity, close LPPosition PDA
- Emit LiquidityWithdrawn event

6. Settlement Program: Gaussian Kernel Payouts

6.1 Payout Function

At market resolution, payouts follow a Gaussian kernel centered on the TWAP-resolved outcome:

payout(p) = e^(−(actual − predicted)² / (2σ²))

For range positions [a, b], the integrated payout is:

payout([a,b]) = (σ√(π/2) / (b−a)) × [erf((b−actual)/σ√2) − erf((a−actual)/σ√2)]

On-Chain erf() Implementation

Uses the Abramowitz & Stegun polynomial approximation (formula 7.1.26):

erf(x) ≈ 1 − (a1·t + a2·t² + a3·t³ + a4·t⁴ + a5·t⁵) × e^(−x²)

where t = 1 / (1 + 0.3275911 × |x|)

5th-order polynomial achieves |error| < 1.5×10^{-7}. All computation in fixed-point u128 with 10^{12} scaling.

6.2 resolve_market

- Assert current_slot ≥ market.resolution_timestamp
- Read TWAP from stored price samples over twap_window_slots
- Apply outlier rejection: discard samples > 3σ from running median
- Compute final TWAP value
- Set market.resolved_outcome = TWAP, market.status = Settled
- Emit MarketResolved event

6.3 claim_payout

- Read position’s range [range_low, range_high] and shares
- Compute Gaussian payout using erf() approximation
- gross_payout = payout_ratio × shares × max_payout_per_share
- Deduct 0.1% settlement fee
- Transfer net payout from vault to position owner
- Close Position PDA

6.4 TWAP Price Sampling

The protocol accumulates price samples during the TWAP window via a keeper bot that calls sample_price every N slots. Each sample is stored in a ring buffer PDA:

Field	Type	Description
market	Pubkey	Market account

samples	[u64; 128]	Ring buffer of price samples (scaled 10^6)
sample_count	u16	Number of samples recorded
write_index	u16	Current write position in ring buffer
bump	u8	PDA bump

7. Liquidation Program: Dutch Auction Engine

7.1 Value-at-Risk (VaR) Model

VaR_α = μ_portfolio − z_α × σ_portfolio

At 99% confidence (α = 0.99, z = 2.326), the protocol estimates maximum loss. When position margin < VaR, liquidation triggers.

Margin Health Computation

margin_ratio = margin_deposited / (entry_cost × leverage / 100)

health = margin_ratio / required_margin_ratio

When health < 1.0, the position is marked Liquidatable.

7.2 trigger_liquidation

- Anyone can call this instruction for an unhealthy position (permissionless)
- Validates margin_ratio < threshold
- Creates Auction PDA with start_price = 105% of estimated position value
- Sets position.status = Liquidatable
- Emits LiquidationTriggered event

7.3 claim_auction

- Any address can claim during an active auction
- current_price = start_price − (decay_rate_bps × (current_slot − start_slot) × start_price / 10000)
- Claimer pays current_price in USDC
- Position ownership transfers to claimer
- Remaining margin returned to original owner
- Emits AuctionClaimed event

7.4 Auction Expiry

If no one claims after max_auction_slots (~2700 = 18 minutes), the protocol closes the position at market value and returns residual to the owner.

8. Oracle Integration

8.1 Primary: Pyth Network

- Uses Pyth pull oracle model on Solana
- Reads price, confidence interval, and expo fields
- Rejects prices where confidence > 2% of price (too uncertain)

8.2 Secondary: Switchboard

- Fallback if Pyth feed is stale (> 30 seconds old)
- Same validation: confidence check, staleness check

8.3 TWAP Guards

- Settlement uses 30-minute TWAP, not a single price snapshot
- Outlier rejection: discard samples > 3σ from running median
- Minimum 10 samples required for valid settlement

8.4 Optimistic Oracle (Event Markets)

For subjective/event-based markets (not price feeds):

- Proposer submits outcome with a bond (e.g., 100 USDC)
- Dispute window: 24 hours
- If disputed, resolution escalates to staked resolver committee
- Bond is slashed if proposal is overturned

9. Fee Collection & Insurance Fund

9.1 Fee Schedule

Fee Type	Rate	Destination	When Applied
Trade fee	0.3%	70% to LPs, 30% to protocol treasury	Position entry and exit
Settlement fee	0.1%	Protocol treasury	Payout claims
Leverage fee	0.05%/slot	Protocol treasury	Per-slot on leveraged positions

9.2 Insurance Fund

A PDA-controlled USDC reserve that covers LP impermanent loss beyond a 5% threshold.

Seeds: [b"insurance_fund"]

Funding Sources

- External: $ISO creator fees from PumpFun (deposited via admin instruction)
- Internal: A portion of protocol treasury fees (configurable by governance)

Payout Logic

- Computed automatically during withdraw_liquidity
- IL = max(0, deposit_amount − current_value)
- If IL > 5% of deposit_amount: insurance_payout = IL − (0.05 × deposit_amount)
- Deducted from insurance_fund_balance atomically in the same transaction

Depletion Safeguards

- If fund < minimum_reserve_ratio: pause new LP deposits to high-risk ranges
- Adjust IL threshold upward (e.g., 5% → 8%) until fund recovers

10. Instruction Reference

iso_core

Instruction	Signer	Key Accounts	Description
initialize_market	Authority	Market, MarketConfig	Create a new prediction market with bins
open_position	Trader	Market, Bins[], Position, Vault, UserUSDC	Buy shares in a range
close_position	Trader	Market, Bins[], Position, Vault, UserUSDC	Sell shares and receive USDC
update_market_config	Authority	Market, MarketConfig	Modify market parameters (admin only)

iso_vault

Instruction	Signer	Key Accounts	Description
deposit_liquidity	LP	Market, Vault, LPPosition, Bins[], UserUSDC	Deposit USDC and select range
withdraw_liquidity	LP	Market, Vault, LPPosition, Bins[], InsuranceFund, UserUSDC	Withdraw with IL protection
claim_fees	LP	LPPosition, Vault, UserUSDC	Claim accumulated trading fees
fund_insurance	Admin	InsuranceFund, AdminUSDC	Deposit external funds to insurance

iso_settlement

Instruction	Signer	Key Accounts	Description
sample_price	Keeper	Market, TwapBuffer, OracleFeed	Record a TWAP price sample
resolve_market	Anyone	Market, TwapBuffer	Finalize market with TWAP outcome
claim_payout	Trader	Market, Position, Vault, UserUSDC	Claim Gaussian payout after settlement
settle_lp	LP	Market, LPPosition, Vault, InsuranceFund, UserUSDC	Settle LP position after resolution

iso_liquidation

Instruction	Signer	Key Accounts	Description
trigger_liquidation	Anyone	Position, Auction, Market	Start Dutch auction for unhealthy position
claim_auction	Claimer	Auction, Position, Vault, ClaimerUSDC	Claim position at current auction price
expire_auction	Anyone	Auction, Position, Vault, OwnerUSDC	Close expired auction, return residual

11. Error Codes

Code	Name	Description
6000	MarketNotActive	Market is not in Active status
6001	MarketAlreadySettled	Market has already been resolved
6002	InvalidRange	range_low ≥ range_high or out of market bounds
6003	SlippageExceeded	Cost exceeds max_cost parameter
6004	InsufficientBalance	User USDC balance < required amount
6005	InsufficientLiquidity	Vault has insufficient unreserved USDC
6006	PositionNotLiquidatable	Margin health is above threshold
6007	AuctionExpired	Auction has passed max_auction_slots
6008	AuctionNotActive	Auction already claimed or expired
6009	OracleStale	Price feed is older than max staleness
6010	OracleConfidenceTooWide	Oracle confidence > 2% of price
6011	TwapInsufficient	Fewer than 10 TWAP samples recorded
6012	ResolutionTooEarly	Current time < resolution_timestamp
6013	InsuranceFundDepleted	Insurance fund below minimum reserve
6014	Unauthorized	Signer is not the required authority
6015	MathOverflow	Arithmetic overflow in fixed-point computation

12. Security Model

Isometric is designed with multiple layers of protection to safeguard user funds, ensure fair market resolution, and prevent economic exploitation.

Oracle Safety

The protocol never relies on a single price snapshot for any critical operation. All settlement uses Time-Weighted Average Prices (TWAP) computed over a 30-minute window, making flash-loan or single-block price manipulation economically infeasible.

- Dual oracle stack: Pyth Network (primary) with Switchboard (automatic fallback)
- Staleness rejection: feeds older than 30 seconds are automatically rejected
- Confidence filtering: prices with uncertainty > 2% are discarded
- Outlier rejection: TWAP samples deviating > 3σ from running median are excluded
- Minimum sample threshold: settlement requires at least 10 valid price samples

TWAP Protection

Binary prediction markets resolve at a single timestamp, creating a concentrated attack surface. Isometric's 30-minute TWAP window distributes the resolution across hundreds of price samples, requiring sustained market manipulation rather than a single-block exploit.

- Ring buffer stores up to 128 price samples during the settlement window
- Statistical outlier rejection prevents individual manipulated samples from affecting the outcome
- The cost of sustained manipulation across a 30-minute window on a liquid market is prohibitively high

Dutch Auction MEV Resistance

Traditional DeFi liquidations are vulnerable to MEV extraction — bots front-run liquidation transactions to extract value. Isometric's Dutch auction mechanism naturally resists this:

- Descending price means there's no "optimal" moment to front-run — waiting always yields a better price
- The auction finds its own market-clearing price through natural competition
- Original position holders receive residual margin, not a fixed penalty
- Auction expiry (~18 minutes) ensures positions are always resolved, preventing stuck capital

Insurance Fund Safeguards

The Isometric Insurance Fund operates with built-in circuit breakers to prevent catastrophic depletion:

- 5% IL threshold means the fund only covers extreme losses, not normal market movements

- Dual funding sources (protocol fees + $ISO creator fees) provide independent revenue streams
- Minimum reserve ratio monitoring: if fund drops below threshold, new deposits to high-risk ranges are paused
- Dynamic threshold adjustment: IL coverage threshold increases (e.g., 5% → 8%) during low-reserve periods
- All insurance payouts are atomic — computed and distributed in a single transaction to prevent partial state

Slippage Protection

Every position entry includes a max_cost parameter. If market conditions change between transaction submission and execution (e.g., due to other trades), the transaction reverts rather than executing at an unexpected price. This prevents sandwich attacks and unexpected costs.

13. Security Considerations

Account Validation

- All PDAs are verified against expected seeds using Anchor constraints
- Every instruction validates signer authority
- Token accounts are verified as owned by expected PDA

Arithmetic Safety

- Checked arithmetic everywhere (checked_add, checked_mul, checked_div)
- Fixed-point math with clear scaling constants (SCALE_6 = 1,000,000, SCALE_12 = 1,000,000,000,000)
- exp() and ln() handle edge cases (very large/small inputs)
- Tested with extreme values: 0 liquidity, max u64 shares, etc.

Oracle Security

- Never use a single price snapshot for settlement
- Validate oracle account ownership (Pyth program ID)
- Check price publish time for staleness
- Reject prices with wide confidence intervals

Reentrancy & CPI

- Anchor’s constraints handle most reentrancy protections
- CPI calls are verified to not create circular dependencies
- Settlement and liquidation are atomic (no partial state updates)

Economic Attack Prevention

- Sandwich protection: max_cost slippage parameter on open_position
- TWAP protects against flash-loan manipulation at settlement
- Dutch auction prevents MEV extraction during liquidation
- Insurance fund depletion safeguards prevent unbounded liabilities

14. Testing Requirements

14.1 Unit Tests

- LMSR cost function: verify cost increases with shares, decreases with liquidity
- Gaussian payout: verify erf() approximation accuracy against known values
- Sigmoid pricing: verify bounded output, correct inflection behavior
- Fixed-point math: test overflow edge cases with max values
- Bin overlap computation: test partial overlaps, exact matches, no overlap

14.2 Integration Tests

- Full lifecycle: create market → deposit LP → open position → resolve → claim payout
- Liquidation flow: open leveraged position → price moves against → trigger → auction → claim
- Insurance: LP deposits → outcome in range → IL > 5% → insurance payout correct
- Multi-user: multiple LPs and traders in same market, verify fee distribution
- Edge cases: empty bins, single-bin ranges, maximum bin_count

14.3 Fuzzing

- Use trident or custom fuzz harness on LMSR cost function
- Fuzz position entry/exit with random ranges, shares, and liquidity distributions
- Verify no instruction sequence can drain vault below reserved amount

15. Roadmap & Deployment

Devnet

- Deploy all 4 programs to Solana devnet
- Initialize test markets with Pyth devnet price feeds
- Community testing with devnet tokens
- Full integration testing against deployed programs
- SDK generation for frontend and third-party integrations
- Set up keeper bot for TWAP price sampling

Mainnet-Beta

- Comprehensive security audit by a reputable firm
- Deploy with upgrade authority held by multisig
- Initialize insurance fund with seed capital
- Configure production Pyth/Switchboard feeds
- Set conservative initial parameters (low leverage caps, wide σ)
- Progressive rollout: limited markets, capped deposits, monitoring period

16. Technical Architecture

Isometric is built on Solana using the Anchor framework. The protocol is decomposed into four independent, composable on-chain programs that communicate via Cross-Program Invocation (CPI). This section provides a high-level summary of the on-chain program structure.

Why Anchor

Anchor is the industry standard for Solana DeFi. It auto-generates account security checks and TypeScript SDKs from program code, eliminating entire categories of exploits. Every major protocol in the ecosystem (Meteora, Jupiter, Drift) uses it, ensuring the deepest pool of auditor and developer talent. For Isometric's complex fixed-point math (LMSR cost functions, Gaussian erf() approximations), Anchor provides the safety guarantees that alternatives cannot match at the same development velocity.

Program Summary

Program	Role	Core Math
iso_core	LMSR pricing engine, market creation, position management	Range-weighted cost function, sigmoid bonding curves, bin overlap computation
iso_vault	USDC custody, LP	Proportional liquidity distribution, IL

	deposits/withdrawals, fee distribution	computation, insurance payout logic
iso_settlement	Market resolution, Gaussian payouts, TWAP finalization	erf() polynomial approximation (Abramowitz & Stegun), fixed-point u128 arithmetic
iso_liquidation	Margin monitoring, Dutch auction execution	VaR estimation, descending price computation, margin health calculation

Design Principles

- Modularity: Four separate programs that can be upgraded independently
- Determinism: All math in fixed-point u128 with 10^{12} scaling — no floating point
- Composability: Every account is a PDA with deterministic seeds — third parties can build on top
- Permissionless: Liquidation triggers, market resolution, and TWAP sampling are callable by anyone
- Atomicity: All critical operations (settlement, insurance payouts, liquidation) execute in a single transaction

Fixed-Point Arithmetic

Solana programs cannot use floating-point operations (they're non-deterministic across validators). Isometric uses two precision levels:

- SCALE_6 (10^6 = 1,000,000): Used for USDC amounts, prices, and user-facing values
- SCALE_12 (10^{12} = 1,000,000,000,000): Used for intermediate math (LMSR cost, erf() computation)

All exp() and ln() functions are implemented as lookup tables with linear interpolation, bounded to prevent overflow. The erf() implementation uses a 5th-order polynomial that achieves $|\text{error}| < 1.5 \times 10^{-7}$.

Event Emission

Every state-changing instruction emits a structured event for off-chain indexing:

- PositionOpened / PositionClosed — trade tracking and P&L computation
- LiquidityDeposited / LiquidityWithdrawn — LP activity and TVL tracking
- MarketResolved — settlement outcome and TWAP value
- LiquidationTriggered / AuctionClaimed — risk event monitoring
- InsurancePayoutIssued — fund depletion tracking

These events enable real-time dashboards, analytics, and third-party integrations without polling on-chain state.

— End of Technical Overview —

Follow us on X: @isometricmkts