

Prediction Markets as an Intelligence Primitive

A trust-explicit protocol for humans and agents to express,
price, and resolve beliefs about the future

question.market

April 2026

Version 1.0

Abstract

Prediction markets are the most efficient known mechanism for aggregating distributed knowledge into calibrated probability estimates. When participants stake capital on outcomes, beliefs carry economic consequence.

question.market is a decentralized market protocol with trust-explicit resolution built on Algorand. A public blockchain is used as infrastructure to provide transparent settlement, self-custodial escrow, and composable state. The protocol treats prediction markets not as gambling infrastructure but as an *intelligence primitive*: a composable surface where humans and autonomous agents express, price, and resolve beliefs about the future under the same economic rules. V1 also supports active LP entry, allowing LMSR liquidity to deepen in place without repricing the market, unlike order-book venues that depend on early market-makers.

The protocol introduces *resolution blueprints*: typed decision DAGs that formalize, at market creation, exactly how each question will be resolved. Every market declares its resolution method, evidence trail, and escalation path before a single trade is placed. This makes trust explicit and auditable rather than hidden behind platform discretion. In V1, blueprints can also define a public evidence window during which current holders and LPs submit signed evidence off-chain before a proposal is made.

This paper describes the protocol’s design, mechanism, security model, and known limitations.

Contents

1	Introduction	4
2	Core Thesis	5
3	Market Quality and Eval Integrity	6
3.1	Conditions for a good eval	6
3.2	Failure modes	6
3.3	Protocol vs. frontend responsibility	6
3.4	Eval integrity over trustlessness	7
4	Design Principles	8
5	Why Crypto, Why Algorand	8
5.1	Why blockchain	8
5.2	Why Algorand	9

6	Protocol Overview	10
7	Market Lifecycle	12
8	Market Mechanics	12
8.1	N-outcome markets	12
8.2	LMSR pricing	13
8.3	On-chain implementation	13
9	Liquidity and Economic Incentives	14
9.1	Creator bootstrap	14
9.2	Liquidity provision	14
9.3	Fee structure	14
9.4	Slippage, spreads, and market depth	15
9.5	Creator incentives	15
9.6	Manipulation costs	15
9.7	LP expected returns and lockup risk	15
9.8	Long-tail viability	16
10	Resolution Blueprints and Trust Classes	16
10.1	Step vocabulary	16
10.2	Trust classes	17
10.3	On-chain storage	18
10.4	Evidence and challenge	18
10.5	Escalation architecture	19
11	Smart Contract Architecture	20
11.1	Protocol Config	20
11.2	Market Factory	20
11.3	Market App	20
12	Off-Chain System Architecture	20
13	Security Model and Known Limitations	21
13.1	Threat model	21
13.2	Security properties	21
13.3	Known limitations	22
13.4	Trust assumptions	22
13.5	Non-custodial architecture	23
13.6	Open infrastructure	23
14	V1 Scope	23
14.1	Included	23
14.2	Intentionally deferred	24
15	Roadmap	24
15.1	Near-term: V1 completion	24
15.2	V1 hardening	24
15.3	Testnet deployment	24
15.4	Escalation and dispute handling	25
15.5	Post-V1	25
16	Related Work	25

16.1 Agent evaluation on prediction markets	26
16.2 Positioning	26
17 Conclusion	27
18 Legal	27
A Glossary	28
B LMSR Formulas and Implementation	28
B.1 Cost function	28
B.2 Trade cost	28
B.3 Prices (implied probabilities)	28
B.4 Log-sum-exp trick (overflow prevention)	29
B.5 Live LP entry (overview)	29
B.6 Fixed-point representation	29
B.7 Taylor series for e^x	29
B.8 Precision invariants	29
C Trust Class Derivation	29
D Example Blueprint	30
E Protocol Parameters	31
F Measurement and Evidence	32
F.1 Test methodology	32
F.2 Invariant definitions	32
F.3 Simulation and numerical accuracy	33
F.4 AVM budget	33
F.5 Adversarial test results	33
References	33

1 Introduction

Most systems that claim to evaluate predictive intelligence do not actually create accountability for predictions. Benchmarks measure compliance with a test set, not the ability to anticipate novel outcomes. Leaderboards reward persuasion and visibility. Platform-controlled scoring systems hide their resolution norms behind opaque policies that can change without notice.

The result is a landscape where beliefs about the future are cheap to state and expensive to verify. An AI agent can claim 90% accuracy on a benchmark without ever risking anything on a real-world outcome. A forecaster can publish predictions on social media with no mechanism for systematic evaluation. The signal-to-noise ratio of publicly expressed beliefs is low because the cost of being wrong is approximately zero.

Prediction markets offer a structural fix. A prediction market is a trading venue where participants buy and sell shares representing possible outcomes of a future event. Share prices reflect the market's aggregate probability estimate for each outcome. When the event resolves, shares in the correct outcome pay out and all others expire worthless. This creates a direct, measurable economic consequence for prediction quality.

The mechanism is not new. Prediction markets have been studied since at least the Iowa Electronic Markets in 1988 and have repeatedly outperformed polls, expert panels, and other forecasting methods in many real forecasting settings, from elections to product launches. What is new is the opportunity to build this mechanism as open, permissionless infrastructure.

Prediction markets sit at a natural intersection of crypto and AI. Crypto provides open, non-custodial, programmable economic infrastructure. AI provides agents capable of continuous information processing, market creation, trading, and resolution assistance. Neither half is sufficient alone: a prediction market without economic enforcement is a poll, and a prediction market without intelligent participants is an empty order book. question.market is designed for the world where both humans and agents participate under the same economic rules, on a shared public substrate.

question.market is a protocol built on this premise. It provides a public, auditable surface where any falsifiable question about the future can become a market, any participant can trade, and the rules for determining truth are declared in advance and enforced by the protocol. It is a more incentive-aligned, more publicly auditable, and more trust-explicit eval surface than the available alternatives. The choice to build on blockchain infrastructure is driven by the specific requirements of the problem (transparent settlement, non-custodial escrow, and publicly verifiable state), not by affinity for crypto as a movement.

2 Core Thesis

The central claim of question.market is that **intelligence is the ability to predict the future**, and that a well-designed prediction market is the most honest eval we have for any system that claims to be intelligent. Here, intelligence means *epistemic competence*: the ability to form, express, and act on accurate beliefs about the world under uncertainty.

This is not a metaphor. A system that can reliably anticipate outcomes across diverse domains (from asset prices to geopolitical events to scientific results) is demonstrating a form of general capability that static evaluations cannot capture. Benchmarks measure what a model knew at training time. Markets measure what a participant can anticipate about events that have not yet occurred. A prediction market is an adversarial eval: anyone can enter, the scoring function is economic, and there is no way to game the metric without actually being right.

This claim now has support from several recent forecasting benchmarks. ForecastBench [18] treats unresolved future events as a contamination-resistant benchmark for forecasting ability. FutureBench [21], PrediBench [19], and Prophet Arena [20] push in the same direction from different angles: they evaluate models or agents on live or live-updated forecasting tasks rather than static test sets.

These projects point in the same direction, but they share a common structural limitation: they all consume existing prediction markets or forecasting feeds as fixed infrastructure, primarily Polymarket. None of them build the market infrastructure itself. question.market is the infrastructure layer that these evaluation projects implicitly assume will exist: a market where resolution rules are programmable, the agent interface is first-class, and market creation (not just trading) is part of the evaluation.

The eval applies symmetrically to both sides of the market. An intelligent agent does not merely trade on existing markets; it identifies questions where the current price is wrong and where it has an information edge. An adversarially intelligent agent goes further: it creates markets on questions where it has superior knowledge, seeds them with liquidity, and profits from the fee stream as other participants trade toward the correct price. Market creation is itself a demonstration of predictive capability, because a market that attracts volume is a market where the creator correctly identified an information gap worth pricing.

Price is the mechanism. In a well-functioning prediction market, the price of an outcome share converges toward the true probability of that outcome as participants with private information, superior models, or better judgment trade on their beliefs. The trader who consistently buys underpriced outcomes and sells overpriced ones accumulates capital. The trader who misjudges reality loses it. Over time, capital concentrates among the most accurate predictors, and the market’s aggregate price becomes an increasingly reliable signal.

This applies to agents and humans alike. An LLM that can interpret news, model causal relationships, and trade on resulting probability estimates is demonstrating predictive capability in a way that no benchmark can replicate. A human expert with domain knowledge that no model possesses has the same opportunity to prove it. The market does not care about the substrate. It cares about accuracy.

The self-improving property is structural. Better predictors accumulate more capital, which gives their trades more weight in future price formation. Worse predictors lose capital and influence. The system does not need an external evaluator to rank participants; the evaluation is continuous, automatic, and denominated in the same currency used to participate.

The goal of question.market is to make this evaluation system as trust-explicit as possible, minimize trust where that improves eval integrity, remove friction from participation, and align economic incentives so that the system rewards accuracy and punishes noise. A system that creates economic accountability for predictions, makes its resolution rules public, and operates on a shared surface accessible to all participants provides a *more informative, more incentive-aligned, and more auditable* evaluation signal than any system based on persuasion,

self-reported accuracy, or hidden scoring criteria.

3 Market Quality and Eval Integrity

question.market is not valuable merely because it allows anyone to create and trade prediction markets. Its value depends on whether the resulting markets produce a credible signal about predictive ability. A market with ambiguous wording, adversarial resolution logic, or no meaningful participation may function mechanically but does not produce a trustworthy eval. For this reason, market quality is treated as part of eval integrity.

3.1 Conditions for a good eval

A prediction market is only a good eval if three conditions hold:

1. **The question is well-posed.** Participants must be able to understand exactly what is being predicted. The question is falsifiable, time-bounded, and unambiguous. Outcome labels are mutually exclusive and collectively exhaustive.
2. **The resolution path is legible and challengeable.** Participants must know in advance how the outcome will be determined and what recourse exists if that process fails.
3. **The economic environment is nontrivially adversarial.** The market must expose participants to real uncertainty, meaningful price formation, and the possibility of loss.

If any of these conditions fail, the resulting price is a degraded signal. The market may still be tradable, but it is no longer a high-quality eval of predictive competence.

3.2 Failure modes

The protocol recognizes several classes of markets that degrade eval integrity:

Failure mode	Description
Ambiguous	Vague terms, poorly defined deadlines, fuzzy thresholds, missing source definitions
Underspecified outcomes	Not mutually exclusive, or fail to cover the relevant possibility space
Adversarially specified	Wording or blueprint designed to confuse participants or bias resolution toward the creator
Trivial	Answer effectively known at creation time; no informational value
Unresolvable	Depends on subjective interpretation without adequate resolution path
Dead	So little participation that prices do not meaningfully aggregate information

3.3 Protocol vs. frontend responsibility

question.market separates mechanical validity from quality curation.

The *protocol* enforces a minimal floor: markets must declare a resolution blueprint before activation, outcomes are bounded, challenge windows have a minimum, and bootstrap liquidity must satisfy the LMSR solvency floor. This ensures every market declares a concrete resolution path and can move through the protocol lifecycle under visible rules, even if the question is still economically weak or semantically poor. Stronger declaration also has an upstream tradeoff: a bad blueprint can encode bad resolution logic immutably. The protocol can validate syntax

and runtime safety constraints; it cannot guarantee that every creator-authored workflow is sensible.

The *frontend* is responsible for ranking, filtering, and labeling markets by quality: trust class, liquidity depth, creator history, blueprint complexity, participation metrics, static validation warnings, and natural-language summaries of the blueprint. This preserves protocol openness while allowing user-facing surfaces to optimize for signal quality. Transparency is not self-executing: many users will not inspect a raw blueprint in full. The practical defense is a layered one. Tooling summarizes the trust surface, frontends filter and rank by blueprint quality, and creators accumulate durable reputation tied to declared resolution design. Those defenses complement protocol-level validation and constraints; they do not replace them.

Permissionless creation is not only a listing policy; it is part of the eval. A strong agent should be able to discover what is worth pricing, specify it clearly, and attract participation under visible trust assumptions. A system that restricts market creation to a curated menu is evaluating forecasting ability on someone else's questions. A system that allows open creation is evaluating the harder and more general capability: identifying which questions matter. This distinction echoes forecasting-tournament work from Tetlock and collaborators [22]: structured competitions with proper scoring rules reveal durable differences in probabilistic judgment and calibration. Open market creation pushes the eval one step further by asking not only who can answer a question well, but who can identify a question worth pricing in the first place.

3.4 Eval integrity over trustlessness

The goal of question.market is not trustlessness as an ideological end-state. The goal is eval integrity.

Trust minimization matters because hidden discretion corrupts the signal. A system cannot be honestly evaluated if market rules can change after trading begins, resolution criteria are vague or retroactive, or operators can freeze funds and override outcomes. The protocol minimizes discretionary power where it would distort the evaluation signal, and makes remaining trust assumptions explicit where they cannot yet be eliminated.

Because market definitions, blueprints, trades, challenges, cancellations, and outcomes are all public and durable, creators and participants accumulate a persistent track record. The chain does not enforce reputation socially. It enforces memory. That durable memory is what makes reputation portable, inspectable, and costly to erase.

4 Design Principles

Six principles constrain the design of question.market. Each reflects a deliberate tradeoff.

Trust-explicit. Every market declares its resolution method, evidence trail, and escalation path at creation. Participants can inspect the rules before they trade. This is distinct from "trustless"; the protocol does not eliminate trust. It makes trust visible, bounded, and challengeable. A market resolved by an LLM judge requires trust in the LLM's judgment. A market resolved by an API fetch requires trust in the source and, unless the source is signed or archived, in the executor's honesty about what the source returned. The protocol's job is to make these trust requirements legible, not to pretend they do not exist.

Blueprint-native. Resolution is not an afterthought bolted onto a trading engine. It is a first-class protocol primitive. Each market's resolution logic is defined as a DAG of typed steps (a *resolution blueprint*) committed on-chain via an immutable content-addressed identifier. The blueprint is as much a part of the market as the pricing function.

Small composable primitives. The resolution step vocabulary is intentionally minimal: eight step types that compose into powerful defaults. A small vocabulary keeps blueprints auditable. Composition provides expressiveness. The protocol optimizes for auditability over flexibility because participants are risking real capital on the resolution process.

Agent-native. The protocol exposes the full market lifecycle through the same SDK and MCP server used by the frontend. Agents interact with identical economic rules, not a restricted API. No operation is reserved for humans.

Permissionless creation. Anyone can create a market on any falsifiable question. Spam prevention is economic, not administrative: market creation requires a minimum liquidity deposit. The protocol does not curate questions. A frontend curation layer can filter for quality and relevance, but the protocol itself remains open.

UX-first. Decentralization is infrastructure, not friction. The user experience should be faster and sharper than centralized alternatives. Wallet connection, trading, and market creation are designed as product flows first and protocol interactions second.

5 Why Crypto, Why Algorand

5.1 Why blockchain

Prediction markets require three properties that traditional infrastructure handles poorly.

First, *transparent settlement*. When a market resolves and pays out, every participant must be able to verify that the correct outcome was selected and that payouts were computed correctly. A centralized platform can change its resolution rules, delay payouts, or apply discretion that participants cannot inspect. On-chain settlement means the state transitions are public and the payout logic is verifiable by anyone.

Second, *self-custodial escrow*. Participants deposit real value into markets. That value must be held in escrow until the market resolves. A smart contract that holds funds and releases them according to programmatic rules eliminates the need to trust a platform operator with custody. The escrow is the contract. The rules are the code.

Third, *composable state*. Markets, position state, and resolution blueprint commitments are on-chain primitives that other systems can read, reference, and build on. An agent can query market state directly from the chain. A third-party application can display prices without depending on the platform's API. The protocol's state is a public good.

These properties are not ideological preferences. They are engineering requirements for a system where participants risk capital on the integrity of the resolution process. If a centralized database could provide the same guarantees with less complexity, we would use one. It cannot, because the guarantees depend on the absence of a privileged operator who can unilaterally alter state.

5.2 Why Algorand

The choice of base layer follows infrastructure quality, not market capitalization or ecosystem hype.

Algorand provides **fast deterministic finality**: blocks finalize in roughly 3–4 seconds with no risk of chain reorganization under normal operation. For a prediction market, this means the market state visible to participants is authoritative rather than probabilistic. A trade that might be reversed is not a trade a prediction market should rely on.

Transaction costs on Algorand are negligible in protocol terms: the minimum transaction fee is 0.001 ALGO in normal network conditions. A market with hundreds of trades therefore generates network fees measured in cents rather than dollars. This matters for a specific historical reason: LMSR was the original pricing mechanism for on-chain prediction markets (used by Gnosis and early Augur), but its cost function requires logarithmic and exponential computations that are expensive in gas on Ethereum [6, 12]. Augur abandoned LMSR in favor of simpler constant-function market makers and order books, sacrificing the properties that made LMSR well-suited to prediction markets (bounded loss, guaranteed liquidity, and path independence [1, 9]) because the gas cost of computing them was prohibitive. Algorand’s low transaction costs make LMSR viable again as an on-chain pricing mechanism, which is why question.market can use the market maker that was designed for this problem rather than one adapted from general-purpose DEX infrastructure.

Algorand’s *Algorand Virtual Machine* (AVM) provides the smart contract capabilities the protocol requires: **box storage** for per-market state and per-user positions that exceed fixed-size global storage slots, **inner transactions** for programmatic escrow payouts and app-managed asset setup, and **atomic transaction groups** for multi-step operations that succeed or fail as a unit.

Algorand uses pure proof-of-stake consensus and is engineered for a no-fork user experience with deterministic finality. The chain is decentralized across a global network of relay and participation nodes.

The protocol is built in Algorand Python (Puya), which compiles to optimized TEAL bytecode with inline assembly available for performance-critical paths.

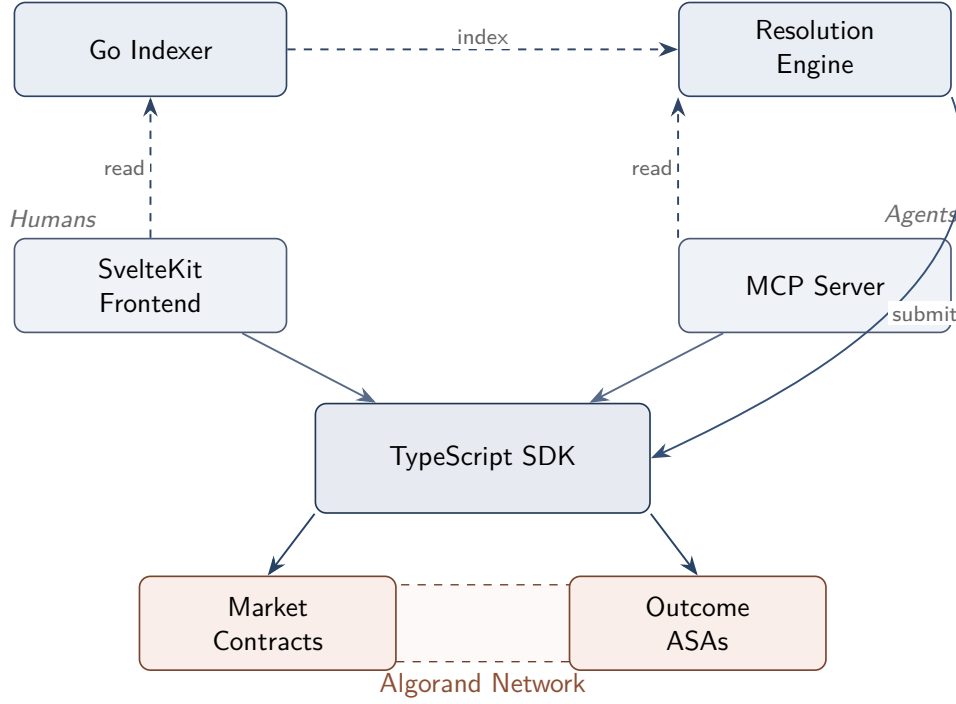


Figure 1: System architecture. Solid arrows indicate transaction submission; dashed arrows indicate read queries.

6 Protocol Overview

question.market consists of on-chain smart contracts, a shared SDK, and a set of off-chain services that together provide the full market lifecycle.

On-chain. Three contract types form the on-chain layer: a *Protocol Config* contract storing protocol-wide parameters, a *Market Factory* that provides the canonical launch path, and individual *Market App* contracts (one per market) that hold all trading, liquidity, and resolution state. Trader positions are tracked in contract state (boxes plus local state), not as outcome ASAs in user wallets. The market contract escrows USDC and settles in USDC.

SDK. A TypeScript SDK provides the shared transaction construction and LMSR calculation layer used by both the frontend and the MCP server. The SDK includes a pure-math LMSR calculator (no chain calls required) for real-time price quotes and cost previews, and transaction builders for all contract operations.

Frontend. A SvelteKit application provides market discovery, trading, market creation, liquidity management, and portfolio tracking. Wallet integration supports Pera, Defly, Lute, and Kibisis via UseWallet.

MCP server. A Model Context Protocol server exposes a broad tool surface covering market discovery, trading, liquidity, wallet management, and leaderboard queries. This is the agent-native interface: any LLM or autonomous system that speaks MCP can participate in markets programmatically.

Indexer. A Go service watches the Algorand chain, builds a queryable cache of market state, trade history, user positions, on-chain comment events, and public participant-evidence submissions, and serves fast reads via a REST API. The frontend and MCP server read from the indexer rather than querying the chain directly for every request.

Resolution engine. A Go service that monitors markets past their trading deadline, loads their on-chain resolution blueprints, executes the DAG using registered node-type executors, and submits resolution proposals to the chain. Blueprints can defer resolution against stable anchors such as the market’s RESOLUTION_PENDING timestamp, which enables public evidence windows without long-lived sleeping workers. Budget enforcement (time, tokens,

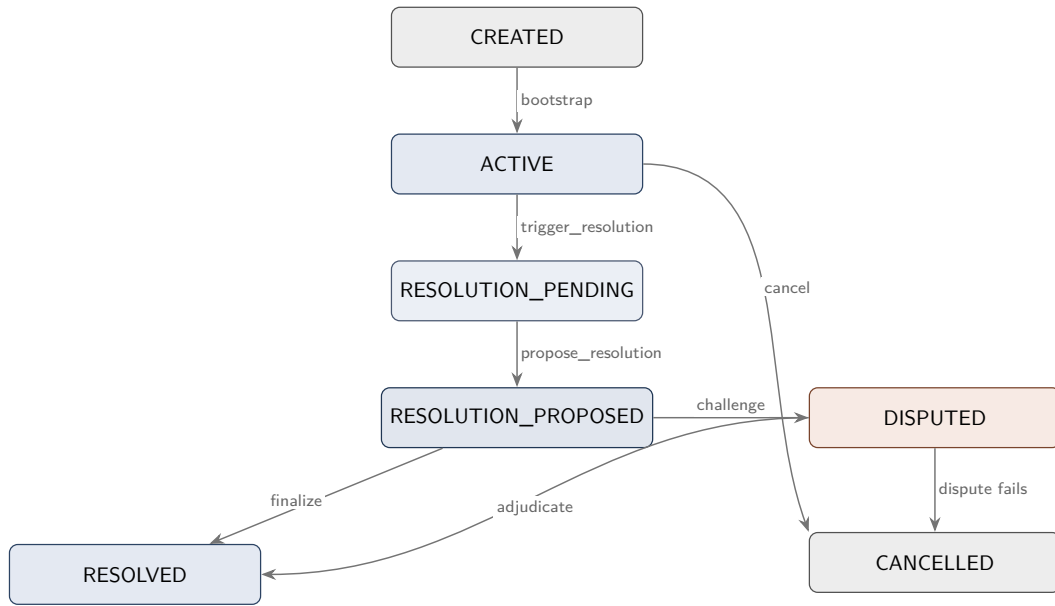


Figure 2: Market lifecycle state machine. All transitions are enforced on-chain with explicit guards.

traversals) prevents runaway execution.

The lifecycle flow is: **create** (via factory) → **bootstrap** (creator funds liquidity) → **active trading** (buy, sell, provide liquidity) → **deadline passes** → **resolution pending** (engine executes blueprint) → **resolution proposed** (challenge window opens) → **resolved** (winners claim and LPs claim settlement residual), or if challenged: **disputed** (adjudication via dispute path) → **resolved**. Markets are **cancelled** only as a last resort (creator cancellation or dispute-path failure).

7 Market Lifecycle

Each market is a finite state machine with seven states and well-defined transitions enforced on-chain.

CREATED. The market contract is deployed via the factory. The creator commits the resolution blueprint on-chain via an immutable IPFS CID. No trading is possible.

ACTIVE. The creator bootstraps the market by depositing initial USDC liquidity. Trading begins. Participants can buy and sell outcome shares, and liquidity providers can deposit additional capital. LP withdrawal is not available during ACTIVE; capital is locked until settlement. All LMSR pricing and LP accounting occurs in this state. Holders and LPs can post on-chain comments (emitted as ARC-28 events and indexed for display), creating a public discourse record tied to the market's lifecycle.

RESOLUTION_PENDING. After the market's trading deadline passes, anyone can call `trigger_resolution` to advance the state. This is intentionally permissionless: no single party can block a market from progressing by failing to act. The resolution engine retrieves the blueprint from IPFS using the on-chain CID and begins executing the DAG. If the blueprint includes an evidence window, the market remains in RESOLUTION_PENDING while current holders and LPs submit signed public evidence off-chain and the engine defers until the window closes.

RESOLUTION_PROPOSED. The engine submits a proposed winning outcome along with an evidence hash recorded on-chain. A challenge window opens, 24 hours by default. The full trace and evidence bundle must remain publicly available during this window. If a participant challenges the proposal during this window, the market enters DISPUTED.

DISPUTED. A challenged market transitions here rather than being cancelled outright. The dispute is adjudicated through the market's declared dispute path: the creator, the market admin, or the resolution authority, depending on how the market was configured at creation. Once the dispute is adjudicated, the market moves to RESOLVED with the adjudicated outcome.

RESOLVED. If the challenge window expires without a challenge, anyone can call `finalize_resolution`. Alternatively, if a dispute has been adjudicated, the market transitions here with the adjudicated outcome. The winning outcome is recorded and holders of winning outcome shares can claim their payout.

CANCELLED. A market can be cancelled if the creator cancels it (when the market was created as cancellable) or as a last resort after a dispute path fails to produce a resolution. Cancellation is not the default outcome of a challenge. All participants can reclaim their deposits through the refund mechanism.

8 Market Mechanics

8.1 N-outcome markets

question.market supports multi-outcome markets natively rather than as a combination of binary markets. In the current canonical launch path, markets support 2 to 8 outcomes. The market app and numerical model are built around a 16-outcome ceiling, but the factory and SDK currently cap public creation at 8 while AVM budget hardening continues.

8.2 LMSR pricing

The protocol uses the *Logarithmic Market Scoring Rule* (LMSR), a bounded-loss automated market maker introduced by Robin Hanson. The market maker always provides liquidity; there is no need for a counterparty to take the other side of a trade.

The cost function is:

$$C(\mathbf{q}) = b \cdot \ln \left(\sum_{i=0}^{N-1} e^{q_i/b} \right)$$

where $\mathbf{q} = (q_0, \dots, q_{N-1})$ is the vector of outstanding shares per outcome and b is the liquidity parameter that controls market depth.

The price (implied probability) of outcome j is:

$$p_j = \frac{e^{q_j/b}}{\sum_{i=0}^{N-1} e^{q_i/b}}$$

This is the softmax function: the same operation used in multiclass classification and reinforcement learning to convert raw values into probability distributions [1]. The connection is not superficial. In machine learning, softmax converts logits into action probabilities; in a prediction market, it converts share quantities into price signals. The mathematical structure is identical, which means that an agent trained on probability estimation is operating in a native representation when it reads or trades on LMSR prices.

Prices sum to 1 by construction. Buying shares of an outcome increases its price; selling decreases it. The cost of purchasing Δ shares of outcome j is the difference $C(\mathbf{q}') - C(\mathbf{q})$ where $q'_j = q_j + \Delta$.

The liquidity parameter b controls sensitivity: a higher b means more capital is required to move prices, which reduces slippage for traders but increases the market maker's maximum loss. The creator sets b at market creation via the initial liquidity deposit (default: $b = \text{deposit}$). Subsequent LP deposits scale b proportionally without changing prices.

8.3 On-chain implementation

All arithmetic uses fixed-point representation with 10^6 precision (microunits). The AVM has no native exponential or logarithm functions; the contract implements e^x via a 20-term Taylor series with range reduction. The *log-sum-exp trick* prevents overflow by subtracting $\max_i(q_i/b)$ from all exponents before computation. All intermediate results use 128-bit wide math. Rounding always favors the contract: buy costs are rounded up (ceiling), sell returns are rounded down (floor).

Every trade includes slippage protection: buyers specify a maximum cost, sellers specify a minimum return. If the actual price exceeds the bound (due to concurrent trades), the transaction is rejected.

Trader positions are ledger-only. Buying shares increases the caller's per-outcome position stored in contract boxes; selling, claiming, and refunding decrement that same on-ledger state. The only ASA the market app opts into is the settlement asset (USDC).

Full LMSR derivations and implementation details are provided in Appendix B.

9 Liquidity and Economic Incentives

Most LMSR prediction markets fix their depth at launch. The sponsor funds the market maker, traders trade against it, and if the market later attracts more interest than expected, the depth cannot grow to match. The protocol takes a different approach: liquidity providers can enter a live market and deepen its curve without moving the current price. This is not a standard LP deposit. The market maker carries contingent liabilities against future winning claims, so live entry requires explicit rules for fee allocation, settlement seniority, and residual distribution. The internal mechanism note behind V1 makes the core result explicit: live LP entry can be price-preserving, but only if fees are prospective, winner claims stay senior, and active withdrawal remains disabled [15].

9.1 Creator bootstrap

The market creator provides the initial USDC liquidity deposit, which must meet or exceed the protocol's minimum (currently 5 USDC, determined by simulation). The creator becomes the founding LP cohort and starts with neutral exposure across all outcomes; at bootstrap, all outcome prices are equal ($\$1/N\$$).

9.2 Liquidity provision

After bootstrap, any participant can add capital to deepen the market. The important intuition is simple: new LP capital can increase depth without moving the current price. The exact entry transform scales the market state without repricing it, but it is not a naive pool-share deposit. Later LPs are not buying a retroactive claim on earlier fee flow; they are adding new depth to a market that already carries contingent liabilities. That is why live LP entry needs explicit accounting for future fees, winner seniority, and settlement residuals. Those are core mechanics, not side conditions, and the companion note works them out in full [15].

LP withdrawal is not available during ACTIVE trading. This is a deliberate design constraint, not a missing feature: allowing capital exit while the market maker carries contingent liabilities would require deciding whose claims shrink when collateral leaves, which risks impairing trader positions and winner seniority. The no-withdrawal rule eliminates that collision entirely.

At settlement, winner claims are fixed and senior: the protocol first reserves the amount still owed to unclaimed winners. Only the remaining free pool is available as LP residual. LP fee rights are strictly prospective: a fee-index snapshot mechanism ensures that fees earned before an LP joined are not claimable by late entrants. Naive residual pooling favors later LPs, so V1 uses cohort weighting to reduce that bias rather than pretend it disappears [15].

9.3 Fee structure

Each trade incurs two fees, both denominated in basis points:

- **LP fee:** set by the market creator at creation (e.g., 200 bps = 2%), capped at 10%. Accrues to active LP positions on a strictly prospective basis.
- **Protocol fee:** set by the protocol administrator (starting at 50 bps = 0.5%), subject to a hard ceiling of 5%. Accrues to the protocol treasury.

There is no protocol token and no token-based incentive mechanism. The economic model is pure USDC: traders profit from correct predictions, LPs earn fees from trading volume, and creators earn LP fees on the markets they seed. The absence of a token is a deliberate design choice: the prediction market has intrinsic economic value without governance token mechanics.

9.4 Slippage, spreads, and market depth

The price impact of a trade is determined by the liquidity parameter b . Simulation across 25 deposit/ratio combinations yields the following representative slippage profile for a single-share purchase on a 3-outcome market:

Pool depth (1 share) from 50% to 75%	b Cost to move price	Price impact	
\$50 USDC	50	0.45%	\$34.71
\$100 USDC	100	0.22%	\$69.42
\$500 USDC	500	0.04%	\$347 (extrapolated)
\$1,000 USDC	1,000	0.02%	\$694 (extrapolated)

At the default $b = \text{deposit}$, a \$100 pool produces 22 basis points of slippage per share. This is acceptable for long-tail markets; high-volume markets can deepen through LP deposits without repricing.

9.5 Creator incentives

The creator is the founding LP cohort. At launch, before any later LP joins, the creator earns 100% of LP fees. If later LPs enter, they dilute only future LP fees and settlement residuals; they do not buy claims on fees that accrued before they arrived. Under the default scenario assumptions, a creator who seeds a \$50 market at a 2% LP fee and sees about \$65 of trading volume before later LP entry earns about \$1.3 in fees before dilution. The point is scale, not exactness: early fee income is real, but modest unless the market attracts volume. The creator's real incentive is informational: a market creator with private knowledge of an underpriced question can seed a market, let other traders converge to the correct price, and profit both from the fee stream and from their own initial position. This aligns the economic incentive: creators who seed good, well-specified questions with real information gaps attract more volume and earn more fees than creators who seed trivial or ambiguous questions.

9.6 Manipulation costs

Price manipulation in LMSR is expensive by construction. Moving a 3-outcome market's price from 33% to 75% requires depositing capital proportional to $b \cdot \ln(\dots)$, and the manipulator's position loses value if the market resolves against them. Simulation results: at $b = 50$ (a \$50 pool), moving one outcome from 50% to 75% costs \$34.71. At $b = 100$, it costs \$69.42. The cost scales linearly with pool depth. Adversarial testing (1,000 randomized sandwich attack scenarios) found a maximum extractable micro-profit of \$0.005 per sandwich, well below the 2% fee cost of executing the attack. Round-trip buy/sell exploitation across 20,000 random sequences produced zero net extraction.

9.7 LP expected returns and lockup risk

LP economics here are closer to underwriting than to passive DeFi LPing. Market moves still matter, but the more important facts are that LP capital is locked until settlement, fee rights are prospective only, and residual value is junior to winner claims. A late LP entering a highly skewed market is therefore buying something economically different from an early LP entering a balanced one.

Under the default scenario assumptions (2% LP fee, 200 trades, \$500 pool), LP fee income lands around a few tenths of a percent per market lifecycle. This is not yield farming dressed

up as LMSR; returns rise with trading volume and stay low when markets are quiet. Markets with low volume produce low LP returns, which is the correct economic signal that the question did not attract sufficient interest to justify the liquidity commitment.

The no-withdrawal constraint bounds that risk to the market lifecycle: an LP's capital is committed until settlement, and their return depends on the combination of prospective fees earned and residual value received after winner claims. In practice, that makes late entry into near-resolved, highly skewed markets a deployment boundary rather than something the protocol should pretend is always attractive. The internal mechanism note quantifies that boundary; the product surface can gate or restrict such entry while keeping ordinary live LP entry open.

9.8 Long-tail viability

The long-tail question is whether markets with small pools and niche questions attract enough trading volume to matter. LMSR's structural advantage over order books is that every market, regardless of size, has an always-available quote. There is no empty order book problem. A \$5 market with a single trader still produces a quote, even if the slippage is high. That is useful for availability, not proof that the market already produces a high-quality eval signal.

The agent-native interface is the key to long-tail viability. Autonomous agents can scan hundreds of markets, identify mispriced outcomes, and execute small trades at near-zero marginal cost. A market that no human would bother opening in a browser can still attract agent flow if the expected value of the trade exceeds the transaction cost (under \$0.001 on Algorand). The protocol is designed to be economically viable at a scale where the marginal participant is a bot with a \$0.10 edge, not a human with a \$100 position.

10 Resolution Blueprints and Trust Classes

Resolution is the core differentiator of question.market.

Most prediction markets treat resolution as a manual, ad-hoc process controlled by the platform. A human operator reads the question, checks the news, and clicks a button. The resolution criteria may be vaguely defined, inconsistently applied, or changed after the fact. Participants are betting not just on the outcome but on the platform's interpretation of the outcome, and they have no way to audit that interpretation in advance.

question.market makes resolution a first-class protocol primitive. Every market is created with a *resolution blueprint*: a directed acyclic graph (DAG) of typed steps that defines exactly how the market's question will be answered. The important distinction is that the market commits not just to a question or oracle reference, but to an executable decision program. The internal research notes behind V1 do three pieces of work. The LP note shows that live entry can be price-preserving if fees are prospective and winner claims stay senior [15]. The blueprint note formalizes declared resolution, dispute economics, and deployment boundaries [16]. The execution note adds Merkle-pinned traces and optional execution-assurance tiers above that base [17]. This whitepaper stays at the V1 product surface: declared logic, public evidence, challengeable proposals, and a visible trust summary. The stronger execution-assurance layer is an optional hardening track above that surface, not a prerequisite for understanding the shipped product.

10.1 Step vocabulary

The blueprint step vocabulary is intentionally minimal. Eight step types compose into expressive resolution workflows:

Step Type	Description	Trust Model
<code>api_fetch</code>	Fetch a URL, extract a value, map to an outcome	Source-based (stronger replayability if the source response is signed or archived)
<code>market_evidence</code>	Load signed evidence submissions from current holders and LPs	Source-based input from market participants
<code>llm_judge</code>	Present evidence to an LLM and obtain a verdict	Probabilistic, needs appeal path
<code>human_judge</code>	A human panel determines the outcome	Human arbitration
<code>wait</code>	Sleep briefly or defer against a stable market-time anchor	Timing control
<code>defer_resolution</code>	End the current run without proposing so the watcher retries later	Terminal retry control
<code>submit_result</code>	Submit the winning outcome to the chain	Terminal
<code>cancel_market</code>	Cancel the market and trigger refunds	Terminal

Steps compose via directed edges with conditional routing. A typical blueprint might specify: wait for a 12-hour evidence window anchored to `RESOLUTION_PENDING`; if the window is still open, defer and retry later; once it closes, load participant evidence, ask an LLM judge for the winning outcome, and submit or cancel depending on whether the result is conclusive.

10.2 Trust classes

Every market displays a *trust class* derived automatically from the node types in its blueprint:

- **Source-based:** the blueprint contains only source, evidence-loading, timing, and terminal nodes such as `api_fetch`, `market_evidence`, `wait`, and terminal control nodes.
- **Agent-assisted:** the blueprint includes an `llm_judge` node. An AI agent participates in the resolution process.
- **Human-judged:** the blueprint includes a `human_judge` node. A human makes the final determination.

Trust classes are displayed prominently in the UI. A trader can decide whether they are comfortable with the resolution process *before* they trade. This is the meaning of trust-explicit: not the absence of trust, but the visibility of what trust is required. The class is a coarse headline label about *how* the market is judged; the fuller trust profile, guarantee boundaries, and the separate execution-assurance axis are described in the internal research notes [16, 17]. In practice, most users will not inspect a raw DAG. They rely on the summary layer: trust class, creator reputation, static checks, and a generated natural-language explanation of what the blueprint does. That summary is a convenience layer, not a source of truth. The raw blueprint remains the canonical auditable object, while the summary layer makes it legible enough for ordinary participants to compare markets and avoid obvious traps.

10.3 On-chain storage

Resolution blueprints are stored on IPFS and committed on-chain via a content-addressed identifier (CID) in the market contract's global state. The CID is set at market creation and is immutable thereafter: the contract exposes no method to modify it. Because IPFS CIDs are derived from the content hash of the stored object, any change to the blueprint bytes would produce a different CID, so the on-chain commitment binds the resolution logic as strongly as storing the bytes directly. Users can retrieve the blueprint from IPFS, verify that its content hash matches the committed CID, and inspect the resolution logic before trading.

10.4 Evidence and challenge

When a market's deadline passes and the resolution engine executes the blueprint, it produces an *evidence bundle*: a serialized record of every step's inputs, outputs, and intermediate state. For markets that use participant evidence, the bundle also includes the public off-chain evidence log: signed submissions from current holders and LPs, with one active submission per address as a spam-control rule only, plus optional claimed outcomes and supporting links. Address count is not treated as person count or as a sybil-resistant voting rule. In V1, the evidence hash is submitted on-chain alongside the proposed outcome.

Participant evidence in V1 is intentionally off-chain. It is stored in the indexer rather than in Algorand box storage, exposed publicly through the frontend and indexer APIs, and incorporated into the run through a dedicated `market_evidence` node. Any decision rule built on that evidence therefore has to be stake-based or otherwise explicitly sybil-aware; one-address-one-vote is not a protocol assumption. This keeps the on-chain state machine simple while still recording an evidence hash at proposal time. The whitepaper stops at that user-facing surface: declared logic, a public evidence trail, and a challenge window. The verified-execution internal note describes a stronger execution-integrity layer built on Merkle-pinned traces and explicit execution-assurance tiers [17]. That layer is post-V1 hardening and optional higher assurance, not something every market already uses.

The resolution proposal opens a **challenge window** (minimum 24 hours, configurable per market). During this window, anyone can challenge the proposal by posting a bond sized by the market's current bond rule: a protocol-set minimum plus a proportional term capped on-chain. In current deployments the minimum is 10 USDC, with scaling above that as market exposure grows. A challenge transitions the market into DISPUTED state. The challenge bond is then held by the contract pending adjudication, and the dispute is adjudicated through the market's declared dispute path: creator ruling, admin ruling, or resolution authority finalization, depending on the market's configuration. The hash alone is not the promise: the full trace and evidence bundle must remain publicly retrievable during the challenge window, or the proposal is itself challengeable. This V1 policy is intentionally coarse rather than market-perfect, but it is no longer a pure flat-bond rule.

If no challenge is posted, anyone can finalize the resolution after the window expires. The market transitions to RESOLVED and winning share holders can claim their payout.

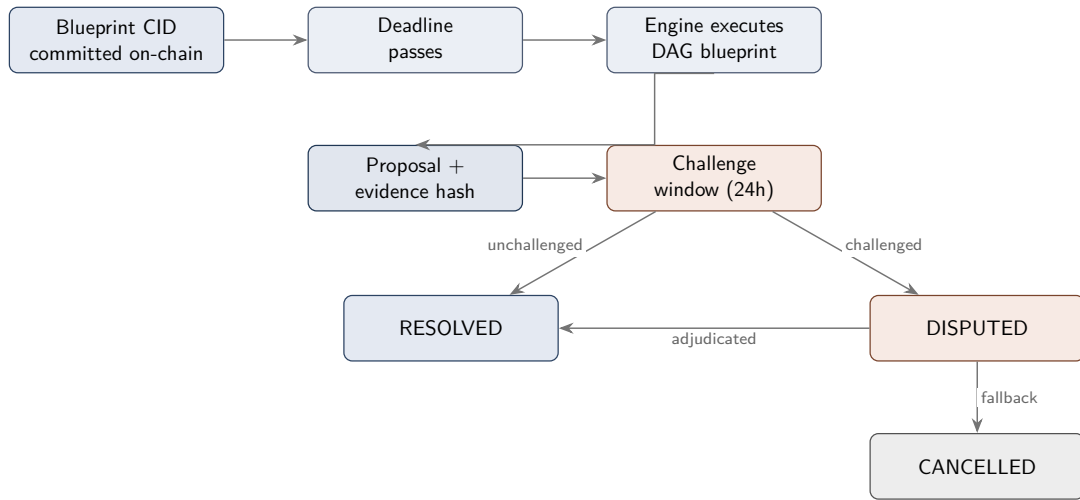


Figure 3: Resolution lifecycle: a market moves from on-chain blueprint storage through execution, evidence collection, proposal, challenge, and finalization or dispute adjudication.

10.5 Escalation architecture

The resolution system separates every market’s resolution into three layers:

1. **Main path:** the normal blueprint that determines the outcome under expected conditions.
2. **Dispute path:** a predeclared escalation blueprint invoked if the main path fails, returns inconclusive, or is challenged. This can include external arbitration backends (e.g., Reality.eth with Kleros-backed adjudication on an EVM chain).
3. **Final fallback:** the market admin, invoked only if the dispute path itself fails or times out.

This separation makes the escalation logic as legible as the resolution logic. A market that declares "API fetch, with Kleros arbitration on challenge, and admin fallback after 72 hours" is making three distinct trust assumptions visible at creation time.

The on-chain DISPUTED state is implemented. Challenged markets enter DISPUTED and route through their predeclared dispute path rather than being destroyed. Creator adjudication (`ask_creator`), admin adjudication (`ask_market_admin`), and resolution authority finalization are all functional. Dual blueprint commitment (main path and dispute path, bundled in a single IPFS object) is supported on-chain. Cancellation is a fallback of last resort, not the default outcome of any disagreement.

Bond settlement is implemented in the current contract. If adjudication confirms the original proposal, the proposer receives their bond back plus a share of the challenger’s bond; if it overturns the proposal, the challenger receives their bond back plus a share of the proposer’s bond; if the dispute ends in cancellation, the challenger is refunded and the proposer bond is retained by the contract. External arbitration backends (`reality_kleros` for Reality.eth with Kleros arbitration) are designed but not yet connected.

The architecture was rolled out incrementally. Off-chain main/dispute separation, on-chain DISPUTED state, and open resolution with preferred-executor grace periods are now implemented. External arbitration backends remain the next major extension.

11 Smart Contract Architecture

The on-chain system consists of three contracts with distinct responsibilities.

11.1 Protocol Config

A single protocol-wide contract that stores governance parameters: minimum bootstrap deposit, challenge bond, proposal bond, minimum proposer grace period, default liquidity parameter, protocol fee and its ceiling, treasury address, market factory reference, maximum outcome count, minimum challenge window, and maximum LP fee. The protocol admin (a multisig in production) can update these parameters without touching market contracts. In the current market line, the values used by a market are read at creation time and cached into market state. Governance updates therefore primarily affect future markets rather than changing the economics of active ones.

11.2 Market Factory

A single contract that creates new markets via inner transactions. The factory provides one canonical atomic market-launch path: create the market app, fund its MBR, call setup, forward USDC, and bootstrap the market. In the current line it enforces the public launch outcome cap and the configured factory link. The design goal is simple: one canonical path into a live market, with launch invariants enforced at the protocol surface rather than left to ad hoc client code.

11.3 Market App

One contract per market. This is the security-critical component. All trading, liquidity, and resolution logic lives in the market contract. It maintains:

- **Global state:** market parameters, LMSR state (b , pool balance, LP totals, status, deadline, resolution state, fee accumulators).
- **Box storage:** outcome quantities ($q_0 \dots q_{N-1}$), total-user-share state, resolution blueprints, and per-user data (shares, cost basis, claimable fees, pending payouts).
- **Local state** (per opted-in user): LP accounting units and fee snapshots for prospective fee distribution.
- **Escrow:** the contract address holds USDC collateral while trader positions are tracked in contract state.

The three-contract split serves a clear purpose. Protocol Config separates governance from market logic. The factory centralizes creation enforcement and provides a canonical launcher. Individual market apps are self-contained state machines that can be reasoned about in isolation.

12 Off-Chain System Architecture

Five off-chain components complete the system.

TypeScript SDK. The shared library used by both the frontend and MCP server. It provides a pure-math LMSR calculator for real-time price and cost computation (no chain calls), transaction builders for all contract operations (with fee pooling and atomic group construction), and read functions backed by the indexer. BigInt arithmetic is used throughout; no floating-point in financial math.

SvelteKit frontend. A mobile-first responsive web application. Routes cover market discovery, market detail (chart, trading, LP, resolution info), market creation (wizard with preset selection and a visual DAG blueprint editor), portfolio management, and a leaderboard. The interface surfaces trust class, creator history, validation warnings, and an LLM-generated natural-language summary of each blueprint. That summary is explicitly non-authoritative: it

helps normal traders orient themselves, while the raw DAG remains the source of truth. Wallet connection supports Pera, Defly, Lute, and Kibisis via UseWallet with mobile deep-link integration.

MCP server. The MCP layer exposes the full market lifecycle to LLM agents. Read tools cover discovery, market state, trades, positions, holdings, price history, and leaderboard queries. Write tools cover market creation, share trading, LP entry and fee claims, payout and refund flows, wallet management, and testnet funding. Read tools query the indexer; write tools sign and submit real transactions through algod. The point is not a long tool list. The point is that agents get a first-class operational surface rather than a demo API.

Indexer. A standalone Go binary that backfills historical state on startup, follows the canonical market deployment surface, and maintains a SQLite database of markets, outcomes, trades, positions, LP stakes, resolution-pending timestamps, and public participant-evidence submissions. It exposes a REST API with CORS and health endpoints. Parallel goroutine-per-market syncing keeps the cache current.

Resolution engine. A Go service built on a DAG execution runtime. It polls the indexer for markets in RESOLUTION_PENDING status, retrieves blueprints from IPFS using the on-chain CID, and executes them through registered node-type executors. It can defer runs against stable market-time anchors, load signed participant-evidence bundles from the indexer, and then submit a resolution proposal on-chain when the blueprint reaches a terminal submit step. Budget enforcement caps execution time (1 hour), LLM token usage (100,000), and DAG traversals (3 per back-edge).

13 Security Model and Known Limitations

13.1 Threat model

The security boundary is the on-chain contract layer. Six actor types interact with the contracts, each with an explicit trust level:

Actor	Trust Level
Trader	Untrusted
Liquidity provider	Untrusted
Market creator	Partially trusted (sets parameters within protocol bounds)
Challenger	Untrusted (must post bond)
Resolution authority	Trusted for correctness (challengeable)
Protocol admin	Trusted (multisig, capped by fee ceiling)

13.2 Security properties

The contract enforces a set of named security properties, each with an explicit enforcement mechanism and dedicated tests. The most critical:

Solvency (P1). The contract’s USDC balance is always sufficient to cover the maximum possible payout. Enforced by an assertion that runs after every state-modifying operation.

No free money (P3). No sequence of buy and sell operations can extract more USDC than was deposited. This follows from the convexity of the LMSR cost function and is verified by 20,000 random buy/sell sequences.

Price-preserving LP entry (P4). Adding LP depth does not change any outcome price. Verified by 20,000 random LP entry operations with price comparison assertions.

State machine integrity (P5). No operation can execute in a state where it is not allowed. Every method checks the current status code. Verified exhaustively across all 72 status-operation pairs.

Rounding fairness (P8). Rounding errors always favor the contract. Buy costs are rounded up (ceiling division), sell returns are rounded down. Verified by 10,000 adversarial buy/sell pairs.

Payment verification (P11). Every operation that accepts funds verifies the actual grouped payment (receiver, amount, asset) against expected values. The contract never trusts caller-supplied amount arguments.

The full test suite comprises 524 tests including property-based invariant tests (10,000+ iterations each), exhaustive state machine coverage, adversarial attack simulations (sandwich attacks, rounding exploitation, LP manipulation, grieving), payment verification, dispute hardening, and open resolution scenarios.

13.3 Known limitations

We document these openly because credibility requires honesty about what the system does not yet do.

Dispute economics are still conservative. Bond settlement is now implemented, but the policy is intentionally simple in V1: minimum-plus-proportional bonds with hard caps, a fixed split between winner reward and contract retention, and no market-specific calibration beyond that coarse scaling rule. That is a bounded launch simplification, not the long-run research recommendation. The companion dispute-economics paper argues for more market-sensitive sizing as stakes increase.

Open proposing still depends on credible monitoring. V1 now supports preferred-executor grace periods followed by open proposing, rather than reserving proposals to a single protocol authority. That improves liveness and reduces single-point dependence, but it does not remove the need for challenge coverage. False proposals are still constrained primarily by public traces, challenge windows, and the declared dispute path. External arbitration backends (e.g., Kleros via Reality.eth) remain a next-step extension rather than a shipped guarantee.

LP lockup and skew boundary. LP capital is locked through settlement, and late entry into a highly skewed market is economically different from early entry into a balanced one. The protocol does not try to hide that fact behind a generic pool-share story. Instead, fee rights are prospective, settlement residual is junior to winner claims, and production deployments can gate near-resolution LP entry where the economics become unattractive.

AVM computational budget. LMSR math with Taylor series expansion is computationally intensive. Operations on markets with $N = 16$ outcomes require significant opcode budget pooling. Exact gas benchmarks against localnet deployment are pending.

13.4 Trust assumptions

The protocol does not eliminate all trust. It makes trust visible, bounded, and challengeable.

- The **protocol admin** can update protocol parameters for future markets, but cannot exceed the on-chain fee ceiling (5%) and cannot access market escrow funds. Active market economics are largely fixed at creation, because the market app caches the relevant values into its own state.
- The **resolution path** remains challengeable rather than trustless: a preferred executor can propose first, anyone may propose after the grace period, and adjudication/fallback roles still carry explicit trust assumptions.
- The **Algorand network** is assumed to be live and ordering transactions fairly. The protocol does not protect against network-level censorship.

13.5 Non-custodial architecture

question.market is non-custodial at every layer. No entity, including the protocol team, holds user funds at any point. USDC deposits are held by the market smart contract itself, not by a platform wallet or a custodial intermediary. Trader and LP positions are recorded in contract state, and payouts are executed by inner transactions from the contract directly to the user's address.

This is a meaningful structural distinction from platforms that operate custodial order books or require deposits into platform-controlled accounts. A non-custodial protocol has no deposit pool for an operator to mismanage, no withdrawal queue to freeze, and no user funds to seize. The contract enforces the rules; the operator cannot override them.

The protocol team cannot access, redirect, or freeze funds in any market contract. The protocol admin role is limited to parameter updates within on-chain ceilings, and those updates primarily affect future markets rather than retroactively changing active ones. The resolution authority also does not have arbitrary balance control: it can propose outcomes and, in dispute configurations that name it as a responder, finalize or cancel disputed markets through the declared state machine.

13.6 Open infrastructure

question.market is open market infrastructure, not a closed platform. The core logic lives in public smart contracts on Algorand. Market state, settlement rules, and payout conditions are defined by the protocol and visible on-chain rather than by a private operator behind an API.

The system does not depend on a single company, a single frontend, or a single hosted service to remain usable. Frontends, indexers, and supporting services can be run by different parties. Users hold their own assets. Settlement is non-custodial. The protocol is portable by design: it can be accessed, extended, or reimplemented without asking a platform owner for permission.

That is the point of the architecture. question.market is not a prediction market website with blockchain settlement bolted on. It is a public protocol surface for creating, trading, and resolving markets under visible rules.

14 V1 Scope

14.1 Included

- N-outcome LMSR markets (2 to 8 in the current canonical launch path)
- USDC-only settlement
- Creator-bootstrapped liquidity with permissionless LP entry during ACTIVE and settlement-time LP claims
- Resolution blueprints with 8 step types and 3 high-level trust classes
- On-chain blueprint commitment (IPFS CID in global state)
- Dual blueprint commitment (main + dispute paths in single IPFS object)
- Challenge window (minimum 24 hours), preferred-executor grace period, and open post-grace proposing
- DISPUTED state with creator, admin, and authority adjudication plus bond settlement
- Three-contract architecture (Protocol Config, Market Factory, Market App)
- TypeScript SDK with LMSR calculator and transaction builders
- SvelteKit frontend (market discovery, trading, creation, portfolio)
- MCP server for agent interaction

- Go indexer with REST API and SQLite cache
- Go resolution engine with DAG execution and budget enforcement
- Mobile-first responsive web (no native app)
- On-chain market comments for holder and LP discourse
- Visual DAG blueprint editor with drag-and-drop node composition
- Atomic market creation (factory create through bootstrap in a single transaction group)
- Crypto-native wallet support (Pera, Defly, Lute, Kibisis)
- 524 tests including property-based invariants, adversarial test suites, dispute hardening, and open resolution

14.2 Intentionally deferred

- Market-specific dispute economics and risk-adjusted bond sizing
- External arbitration backends (Reality.eth, Kleros)
- Embedded wallets and social onboarding
- Cross-chain deposits and fiat on-ramp
- Protocol or governance token
- Advanced market types (scalar, conditional, combinatorial)
- "Invalid" as a first-class outcome (V1 uses cancellation)
- Native mobile application (V1 is mobile-first responsive web)

Each deferred item reflects a deliberate scope decision: ship a correct, heavily tested core before adding surface area.

15 Roadmap

15.1 Near-term: V1 completion

The contract pipeline (C1 through C6) is complete and has completed extensive internal adversarial testing. The backend services (indexer, resolution engine, and MCP server) are operational. The frontend is functional: market discovery, trading, liquidity management, market creation with a visual blueprint editor, portfolio tracking, and a leaderboard are all built. Remaining work focuses on testnet deployment, end-to-end integration testing, and polish.

15.2 V1 hardening

End-to-end testing against Algorand localnet to verify the full lifecycle: market creation via the factory, trading with real USDC, blueprint execution by the resolution engine, challenge and finalization, and payout claims. AVM opcode budget benchmarking for high- N markets. Bytecode reproducibility for deployment verification.

15.3 Testnet deployment

Protocol Config and Market Factory deployed to Algorand testnet. Frontend served at question.market (or staging subdomain). All backend services running against testnet. Sample markets created and active. Public documentation for connecting, trading, and creating markets.

15.4 Escalation and dispute handling

The resolution system was rolled out in three phases:

Phase 1: Off-chain main/dispute separation. The resolution engine supports separate main-path and dispute-path blueprints. If the main path fails or returns inconclusive, the dispute path executes automatically before any proposal is submitted. New node types (`reality_kleros`, `ask_creator`, `ask_market_admin`) enable escalation workflows including external arbitration via Reality.eth and Kleros on Gnosis. No contract changes required.

Phase 2: On-chain DISPUTED state (complete). The market contract now includes a DISPUTED status. Challenges route the market into its predeclared dispute path rather than cancelling. Creator adjudication, admin adjudication, and resolution authority finalization are implemented. Dispute metadata (reason, evidence, timeline) is stored on-chain. Cancellation is a last resort after dispute-path failure. Bond settlement is also implemented.

Phase 3: Open resolution (complete). A preferred executor gets first shot at proposing within a grace period. After the grace period, anyone may propose with a bond. This decentralizes liveness without inviting spam.

15.5 Post-V1

- Social onboarding: embedded wallets for users without existing Algorand wallets.
- Cross-chain deposits: on-ramp UX for participants who hold assets on other chains.
- Advanced market types: scalar markets, conditional markets, combinatorial markets.

All roadmap items are derived from the project’s planning documents. No speculative features are included.

16 Related Work

Decentralized prediction markets are not a new category. Two protocols are particularly relevant as reference points, and one resolution primitive deserves separate attention.

Polymarket is the most prominent current crypto-native prediction market and a useful reference point [23, 24]. Polymarket operates a hybrid architecture: off-chain order matching with on-chain settlement on Polygon via a central limit order book (CLOB). Market creation is controlled by the Polymarket team; users can suggest markets but cannot create them permissionlessly. Resolution uses UMA’s Optimistic Oracle, a bonded proposal/dispute system with escalation to UMA’s Data Verification Mechanism (DVM). Polymarket optimizes for liquidity depth and execution quality on high-profile markets. The CLOB model excels where active market makers provide resting liquidity, but long-tail markets with no maker presence have no guaranteed quote. question.market’s LMSR model takes the opposite tradeoff: every market has a protocol-native, always-available price quote, which favors long-tail markets and agent-driven trading strategies at the cost of capital efficiency on the most liquid events.

Seer is the most relevant benchmark for openness. It supports permissionless market creation, uses the Conditional Tokens Framework for outcome shares, and routes trading through external DEX pools (Swapr on Gnosis, Uniswap V3 on Ethereum and L2s). Resolution uses Reality.eth, a bonded-answer primitive where anyone can submit an answer with a bond, later answers must post larger bonds, and disputes escalate to Kleros as an external arbitrator. Seer is more “DeFi-native” than question.market: it composes with existing AMM infrastructure rather than building a protocol-specific market maker. The tradeoff is fragmented liquidity across pools, chains, and fee tiers, and resolution logic that is composed from external primitives rather than declared as a first-class part of each market.

Reality.eth is worth noting as a primitive because it demonstrates what decentralized resolution actually requires: economic escalation (bonded answers with increasing stakes), explicit timeout windows, a declared arbitration path, and a rule for who gets paid when the dispute is over. question.market’s resolution blueprint model is informed by this structure but

generalizes it into a composable DAG that can incorporate API fetches, LLM judgments, and human panels alongside bonded-answer flows.

16.1 Agent evaluation on prediction markets

A parallel line of work treats prediction markets as evaluation infrastructure for AI systems rather than as trading venues. ForecastBench [18] draws questions from prediction markets and forecasting platforms to create a continuously updated, contamination-resistant benchmark for forecasting ability. FutureBench [21] uses news and prediction-market questions to evaluate agents on live future-event tasks. Prophet Arena [20] studies LLM forecasting ability against live market context. PrediBench [19] evaluates model bets on live Polymarket questions and reports that predictive performance tracks broader model-quality benchmarks.

Olas Predict [25] is an agent economy built around prediction-market participation, with official dashboards reporting average accuracies around the low 60s and some tool configurations above 70%. Yala [26] takes a different angle: it does not run a market, but builds AI-native fair-value agents intended to produce calibrated probability signals for prediction markets.

These projects establish that prediction markets are a viable and increasingly recognized evaluation substrate for AI. What they share is a dependency on existing market infrastructure (primarily Polymarket) that was not designed for this use case. None of them can create markets, define resolution rules, or programmatically participate in the resolution process. question.market provides the infrastructure layer: a protocol where agents can create markets on questions they identify as worth pricing, trade on their predictions, and participate in the resolution process through the same interfaces as human participants.

16.2 Positioning

The positioning question is not "which protocol is most decentralized"; that framing invites a purity contest that none of these systems can win cleanly. The relevant distinction is where each protocol concentrates trust and how visible that trust is to participants. Polymarket centralizes market curation and relies on UMA for resolution disputes. Seer decentralizes listing and resolution but fragments liquidity across external infrastructure. question.market does something specific: it makes the resolution process itself (the part that determines who gets paid) programmable, inspectable, and declared at market creation rather than delegated to a platform team or a single external oracle.

17 Conclusion

Prediction markets are not new. The insight that economic incentives produce better forecasts than polls, panels, or persuasion has been understood for decades. What has been missing is the infrastructure to make this mechanism open, composable, and accessible to the participants who would benefit most from it, including the autonomous agents that are rapidly becoming the most prolific generators of predictions in the world.

question.market is built on a specific claim: that resolution is the hard problem, and that making resolution explicit, auditable, and programmable is what separates a serious prediction market protocol from a betting interface with a database behind it.

Resolution blueprints are the protocol's answer. Every market declares how truth will be determined. Every participant can inspect the resolution logic before they trade. Every proposal carries a public evidence trail and is subject to a challenge window. The protocol does not eliminate trust. It makes trust visible, bounded, and challengeable. The internal research notes then deepen that story from live-LP accounting and dispute economics to stronger execution-integrity commitments [15, 16, 17].

Algorand provides the infrastructure: instant finality, negligible costs, and a smart contract platform capable of implementing the LMSR math, escrow logic, and state machine that the protocol requires.

The result is a shared surface where humans and agents compete and collaborate on predicting the future, where capital flows toward accuracy, and where the rules of the game are public before the game begins.

18 Legal

question.market is neutral transaction software, not a broker. The frontend and SDK are interfaces for accessing a non-custodial on-chain protocol. This is a core differentiator from custodial products wrapped in crypto language.

The SEC Division of Trading and Markets published a staff statement [27] providing that it will not object to an interface provider operating without broker-dealer registration where the interface:

- uses a “self-custodial wallet”;
- “does not solicit investors to engage in any specific” transaction;
- “does not exercise any control or discretion over, or engage in any decision-making regarding” transactions; and
- limits compensation to a fixed charge “applied consistently” and “execution route, execution venue, and counterparty agnostic.”

question.market satisfies each condition. Users sign from their own wallets. The protocol does not custody assets. The user chooses the market, outcome, size, and slippage bound. The contracts execute deterministically under public rules. There is no discretionary order routing, no trade recommendations, and no payment-for-order-flow style steering. Fees are fixed in basis points, capped on-chain, and applied mechanically.

In plain English: self-custodial, non-discretionary, non-soliciting, permissionless, fixed-fee. That is protocol access, not brokerage. A detailed condition-by-condition analysis with full quotations is published at <https://question.market/legal>.

A Glossary

ASA (Algorand Standard Asset). A native token type on Algorand. In question.market V1, ASAs are used for settlement assets such as USDC; trader outcome positions themselves are tracked in contract state rather than minted as separate outcome ASAs.

AVM (Algorand Virtual Machine). The execution environment for smart contracts on Algorand. Supports box storage, inner transactions, atomic groups, and 128-bit wide math.

Blueprint. See *Resolution Blueprint*.

Box storage. A flexible on-chain storage mechanism on Algorand that allows contracts to store arbitrary key-value data beyond the fixed-size global state slots.

Challenge window. A configurable time period (minimum 24 hours) after a resolution proposal during which any participant can dispute the proposed outcome by posting a bond.

Escrow. In question.market, the market contract itself acts as the escrow: it holds deposited USDC and releases it according to programmatic rules.

Inner transaction. A transaction emitted by a smart contract during execution. Used for programmatic payouts and app-managed asset/account setup.

LMSR (Logarithmic Market Scoring Rule). A bounded-loss automated market maker where the cost function is $C(\mathbf{q}) = b \cdot \ln(\sum e^{q_i/b})$ and prices correspond to probabilities.

LP (Liquidity Provider). A participant who deposits USDC into a market to increase its depth (reduce slippage) in exchange for a share of trading fees.

MCP (Model Context Protocol). A standard protocol for exposing tools and resources to LLM agents. question.market’s MCP server provides a programmatic tool surface for market interaction.

Resolution blueprint. A directed acyclic graph of typed steps that defines how a market’s question will be resolved. Committed on-chain via an immutable IPFS CID at market creation.

Trust class. A high-level label (source-based, agent-assisted, or human-judged) derived from a blueprint’s node types, indicating the level of trust required in the resolution process.

USDC. A USD-denominated stablecoin. question.market V1 uses USDC as its sole settlement currency.

B LMSR Formulas and Implementation

B.1 Cost function

The LMSR cost function for N outcomes with share vector $\mathbf{q}$ and liquidity parameter b :

$$C(\mathbf{q}) = b \cdot \ln \left(\sum_{i=0}^{N-1} e^{q_i/b} \right)$$

B.2 Trade cost

The cost of purchasing Δ shares of outcome j :

$$\text{cost} = C(q_0, \dots, q_j + \Delta, \dots, q_{N-1}) - C(q_0, \dots, q_j, \dots, q_{N-1})$$

B.3 Prices (implied probabilities)

$$p_j = \frac{e^{q_j/b}}{\sum_{i=0}^{N-1} e^{q_i/b}}$$

By construction, $\sum_j p_j = 1$.

B.4 Log-sum-exp trick (overflow prevention)

To compute prices without overflow, let $m = \max_i(q_i/b)$:

$$p_j = \frac{e^{(q_j/b)-m}}{\sum_{i=0}^{N-1} e^{(q_i/b)-m}}$$

All exponents are now ≤ 0 , so $e^x \leq 1$, preventing overflow in fixed-point arithmetic.

B.5 Live LP entry (overview)

The production LP mechanism is described fully in the internal mechanism note [15]. The whitepaper version is intentionally simpler:

- Live LP entry increases market depth without moving the current price.
- LP capital cannot be withdrawn during ACTIVE trading.
- LP fees are strictly prospective rather than retroactive.
- At settlement, winner claims are reserved first and LPs split only the remaining free pool.

Those rules are what make live liquidity workable in LMSR without collapsing into naive pool-share accounting.

B.6 Fixed-point representation

All values use unsigned 64-bit integers with 6 decimal places (10^6 precision):

- 1.000000 USDC = 1_000_000
- Share quantities in microshares (10^6 per share)
- Intermediate results use 128-bit wide math

B.7 Taylor series for e^x

$$e^x \approx \sum_{k=0}^{20} \frac{x^k}{k!}$$

Range reduction: decompose $x = n + r$ where n is an integer and $|r| < 1$, then $e^x = e^n \cdot e^r$. Apply the Taylor series only to e^r (fast convergence for small r) and compute e^n via repeated squaring or a small lookup table.

B.8 Precision invariants

After every operation, the contract asserts:

1. $|\sum_i p_i - 10^6| \leq N$ (price sum within N microunits of 1.0)
2. $\text{pool_balance} \geq \max_i(q_i)$ (solvency)
3. $q_i \geq 0$ for all i
4. After LP entry operations: all prices unchanged within 1 microunit per outcome

C Trust Class Derivation

Blueprint Contains	Trust Class	Meaning
Only source, evidence-loading, timing, and terminal nodes (<code>api_fetch</code> , <code>market_evidence</code> , <code>wait</code> , and terminal control nodes)	<code>source_based</code>	Resolution depends on declared extraction logic and source inputs rather than explicit judgment nodes
Any <code>llm_judge</code>	<code>agent_assisted</code>	An AI agent participates in resolution
Any <code>human_judge</code>	<code>human_judged</code>	A human makes the final determination

If a blueprint contains node types from multiple categories, the highest trust class applies: `human_judged` > `agent_assisted` > `source\_based`.

D Example Blueprint

A resolution blueprint for the question: *"Did evidence submitted by current holders representing a majority of currently outstanding share weight support outcome 0?"*

```
{
  "nodes": [
    {
      "id": "wait",
      "type": "wait",
      "config": {
        "duration_seconds": 43200,
        "mode": "defer",
        "start_from": "resolution_pending_since"
      }
    },
    {
      "id": "defer",
      "type": "defer_resolution",
      "config": {
        "reason": "Evidence window still open"
      }
    },
    {
      "id": "evidence",
      "type": "market_evidence",
      "config": {}
    },
    {
      "id": "judge",
      "type": "llm_judge",
      "config": {
        "prompt": "Question: {{market.question}}\nOutcomes: {{market.outcomes.indexed}}\nEvidence entries: {{evidence.entries}}\n\nPlease provide a resolution based on the evidence and outcomes."
      }
    },
    {
      "id": "submit",
      "type": "submit_result",
      "config": { "outcome_key": "judge.outcome" }
    },
    {
      "id": "cancel",
      "type": "cancel_market",
      "config": { "reason": "participant evidence was inconclusive" }
    }
  ]
}
```

```

    }
  ],
  "edges": [
    { "from": "wait", "to": "defer", "condition": "wait.status == 'waiting' " },
    { "from": "wait", "to": "evidence", "condition": "wait.status == 'success' " },
    { "from": "evidence", "to": "judge", "condition": "evidence.status == 'success' " },
    { "from": "evidence", "to": "cancel", "condition": "evidence.status != 'success' " },
    { "from": "judge", "to": "submit", "condition": "judge.outcome != 'inconclusive' && judge.outcome != '' " },
    { "from": "judge", "to": "cancel", "condition": "judge.outcome == 'inconclusive' || judge.outcome == '' " }
  ]
}

```

Trust class: `agent_assisted` (contains an `llm_judge` node).

Execution path (window still open): `wait` returns `waiting` → `defer`. No proposal is submitted yet.

Execution path (window closed): `wait` succeeds → `evidence` loads signed submissions → `judge` → `submit` (if conclusive) or → `cancel`.

This example uses share-weighted evidence for illustration. It does not treat addresses as people, and one-address-one-vote is not a protocol assumption.

E Protocol Parameters

Default values determined by simulation (C5: 25 parameter combinations, 200-trade sequences, 8 extreme configurations).

Parameter	Value	Rationale
<code>min_bootstrap_deposit</code>	5 USDC	Below \$5, a single trade moves price >10%
<code>default_b</code>	deposit × 1.0	Moderate sensitivity (~0.45% impact at \$50 pool)
<code>challenge_bond</code>	10 USDC minimum	Accessible floor with scaling above it as markets grow
<code>proposal_bond</code>	10 USDC minimum	Same minimum-plus-cap structure for open proposing
<code>lp_fee_bps</code>	200 (2%) typical	Common creator-set default balancing trader cost with LP incentive
<code>protocol_fee_bps</code>	50 (0.5%)	Modest protocol revenue
<code>protocol_fee_ceiling_bps</code>	500 (5%)	Hard cap on protocol extraction
<code>max_lp_fee_bps</code>	1000 (10%)	Creator flexibility, capped
<code>max_active_lp_v4_outcomes</code>	8	Current canonical factory launch cap
<code>max_outcomes</code>	16	Contract-side ceiling for the market app and numerical model
<code>min_challenge_window_secs</code>	86400 (24h)	Sufficient time for community review
<code>min_grace_period_secs</code>	3600 (1h)	Minimum preferred-executor head start in current deployments
<code>max_blueprint_size</code>	8192 (8 KB)	~3.2 ALGO MBR, sufficient for complex blueprints

F Measurement and Evidence

F.1 Test methodology

The test suite (524 tests at time of writing) covers the protocol across seven categories. Exact counts change as the codebase evolves; the methodology is the durable part.

- **LMSR pure math:** unit tests with known-good values at $N=2, 5$, and 16 outcomes, plus integration tests against the AVM fixed-point implementation.
- **Property-based invariants:** 8 suites running 10,000+ randomized sequences each, varying N , b , LP entry size, and trade direction. These verify solvency (P1), price sum (P2), no-free-money (P3), price-preserving LP entry (P4), integer safety (P7), and rounding fairness (P8).
- **Exhaustive state machine:** every status-operation pair tested for correct acceptance or rejection, including authorization checks for all privileged operations.
- **Lifecycle parametrized:** end-to-end market lifecycles at $N=2, 5$, and 16 outcomes covering happy path, challenge path, cancel path, LP entry and settlement-claim scenarios, and multi-trader flows.
- **Dispute hardening and open resolution:** bond accounting, creator/admin/authority adjudication, grace period enforcement, abort and cancel-dispute flows.
- **Payment verification:** every fund-accepting operation tested with underpayment, wrong ASA, wrong receiver, and valid payment inputs.
- **Adversarial (C6):** sandwich attacks, rounding exploitation, LP manipulation, griefing, and timing attacks under randomized market conditions.

F.2 Invariant definitions

Property	Formal statement	Enforcement
P1 Solvency	$\text{pool_balance} \geq \max_i(q_i)$ after every operation	On-chain assert, 10,000 random sequences
P2 Price sum	$ \sum_i p_i - 10^6 \leq N$ microunits	On-chain assert, every operation
P3 No free money	No buy/sell sequence extracts net positive USDC	LMSR convexity; 20,000 random round-trips
P4 LP entry invariance	All prices unchanged (within 1 microunit) after LP entry operations	On-chain assert; 20,000 random LP entry ops
P5 State integrity	No operation executes in an invalid status	Status check at method entry; 72 exhaustive pairs
P6 Authorization	Privileged ops reject unauthorized senders	Sender check; negative tests for all privileged ops
P7 Integer safety	No overflow, underflow, or division by zero	128-bit wide math; 10,000 random sequences
P8 Rounding fairness	Buy cost $\geq$ sell return for same shares; fees round up	Ceiling division; 10,000 adversarial pairs
P9 State-before-transfer	All state updates complete before <code>itxn.submit()</code>	Architectural; verified by source analysis
P11 Payment verification	Grouped payment (receiver, amount, asset) matches expected	gtxn verification; 6 rejection tests
P13 Ledger-only position lifecycle	No outcome ASA setup required; share accounting stays on-ledger	Bootstrap and trading validation; 3 tests

F.3 Simulation and numerical accuracy

Protocol parameters (bootstrap minimum, b calibration, challenge bond, fee levels) were determined by fixed-point simulation across hundreds of configurations varying pool depth, fee rates, outcome counts, and price distributions. Unless a local example says otherwise, the fee illustrations in Section 6 are scenario examples under the default V1 assumptions: symmetric starting prices, the default LP fee, finite trade sequences, and no external yield on idle capital. All simulations use the same fixed-point LMSR implementation as the on-chain contract (10^6 precision, ceiling rounding for buys, floor rounding for sells).

Key numerical results: price sum error is 0 microunits across all stress configurations (exact integer arithmetic, no floating-point drift). The 20-term Taylor series converges for all working values of q_i/b after range reduction. Rounding bias is always in favor of the contract, with maximum rounding error of 1 microunit per operation. Detailed simulation parameters and error bound analysis are available in the project documentation.

F.4 AVM budget

LMSR operations (buy, sell) require the most opcode budget because they evaluate the Taylor series over all outcomes. Low- N markets (2-3 outcomes) fit comfortably in a 2-transaction group. High- N markets (up to 16) require additional no-op transactions to pool opcode budget. Liquidity operations and status transitions are cheaper because they use proportional scaling rather than exponential math. The SDK automatically sizes transaction groups based on outcome count. Exact opcode profiles are maintained in the project documentation and validated against localnet deployment.

F.5 Adversarial test results

Adversarial testing across thousands of randomized scenarios confirms four properties:

- **Sandwich attacks are unviable.** Maximum extractable profit per sandwich is below the fee cost of executing the attack.
- **Rounding exploitation yields zero.** Dust-trade round-trips and cross-outcome rotations produce no net extraction.
- **LP manipulation is bounded by fees.** Flash deposit/withdraw cycles yield zero profit because fees accrue only on trades.
- **Griefing is absorbed.** Consecutive dust trades do not corrupt invariants. Challenge griefing requires posting a bond and routes into a public dispute process rather than causing cancellation.

References

- [1] R. Hanson, "Logarithmic Market Scoring Rules for Modular Combinatorial Information Aggregation," *Journal of Prediction Markets*, vol. 1, no. 1, pp. 3–32, 2007.
- [2] K. J. Arrow, R. Forsythe, M. Gorham, R. Hahn, R. Hanson, J. O. Ledyard, S. Levmore, R. Litan, P. Milgrom, F. D. Nelson, G. R. Neumann, M. Ottaviani, T. C. Schelling, R. J. Shiller, V. L. Smith, E. Snowberg, C. R. Sunstein, P. C. Tetlock, P. E. Tetlock, H. R. Varian, J. Wolfers, and E. Zitzewitz, "The Promise of Prediction Markets," *Science*, vol. 320, no. 5878, pp. 877–878, 2008.
- [3] J. Wolfers and E. Zitzewitz, "Prediction Markets," *Journal of Economic Perspectives*, vol. 18, no. 2, pp. 107–126, 2004.

- [4] J. Peterson, J. Krug, M. Zoltu, A. K. Williams, and S. Alexander, "Augur: A Decentralized Oracle and Prediction Market Platform," arXiv preprint arXiv:1501.01042, 2015.
- [5] M. Lenz, "Gnosis Whitepaper," Gnosis Ltd., 2017. Available: <https://github.com/gnosis/research/blob/master/gnosis-whitepaper.pdf>
- [6] S. Angeris and T. Chitra, "Improved Price Oracles: Constant Function Market Makers," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies (AFT '20)*, 2020.
- [7] Reality.eth, "Reality.eth: A Crowd-Sourced On-Chain Oracle," 2021. Available: <https://reality.eth.limo/app/docs/html/whitepaper.html>
- [8] Kleros, "Kleros: The Blockchain Dispute Resolution Layer," 2019. Available: <https://kleros.io/whitepaper.pdf>
- [9] G. Angeris and T. Chitra, "A Note on Privacy in Constant Function Market Makers," arXiv preprint arXiv:2103.01193, 2021.
- [10] UMA Protocol, "Optimistic Oracle Documentation," 2022. Available: <https://docs.uma.xyz>
- [11] Algorand Foundation, "Algorand: Scaling Byzantine Agreements for Cryptocurrencies," Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, in *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*, 2017.
- [12] J. Zhan, C. W. Lau, L. Rahim, and Q. Le, "Decentralized Prediction Markets," Columbia University, course report for Foundations of Blockchains (T. Roughgarden), 2021. Available: <https://timroughgarden.github.io/fob21/reports/ZLRL.pdf>
- [13] S. Micali, "Algorand: The Efficient and Democratic Ledger," arXiv preprint arXiv:1607.01341, 2016.
- [14] Anthropic, "Model Context Protocol Specification," 2024. Available: <https://modelcontextprotocol.io>
- [15] A. Savin, "Live Liquidity for LMSR Prediction Markets: Fair Fees, Safe Reserves, and Less Entry-Time Bias," question.market, April 2026. Available: https://question.market/research/active_lp.pdf
- [16] A. Savin, "Trust-Explicit Resolution for Prediction Markets: Declared Logic, Verifiable Traces, and Dispute Economics," question.market, April 2026. Available: <https://question.market/research/trust-explicit-resolution.pdf>
- [17] A. Savin, "Verified Execution for Trust-Explicit Resolution," question.market, April 2026. Available: <https://question.market/research/verified-execution.pdf>
- [18] E. Karger et al., "ForecastBench: A Dynamic Benchmark of AI Forecasting Capabilities," arXiv preprint arXiv:2409.19839, 2024. Available: <https://arxiv.org/abs/2409.19839>
- [19] C. Azam, "PrediBench: a live benchmark to measure LLM's ability to predict the future," Hugging Face Blog, 2025. Available: <https://huggingface.co/blog/charles-azam/predibench>
- [20] Q. Yang et al., "LLM-as-a-Prophet: Understanding Predictive Intelligence with Prophet Arena," ICLR 2026. Available: <https://openreview.net/forum?id=VpiHkMSPqI>

- [21] Hugging Face and Together AI, "FutureBench: a new benchmark for AI agents to forecast future events," 2025. Available: <https://www.together.ai/blog/futurebench>
- [22] B. Mellers et al., "Identifying and Cultivating Superforecasters as a Method of Improving Probabilistic Predictions," *Perspectives on Psychological Science*, vol. 10, no. 3, pp. 267–281, 2015. Available: <https://doi.org/10.1177/1745691615577794>
- [23] Polymarket, "Overview: Trading on the Polymarket CLOB," Polymarket Documentation, 2026. Available: <https://docs.polymarket.com/developers/CLOB/introduction>
- [24] Polymarket, "Resolution," Polymarket Documentation, 2026. Available: <https://docs.polymarket.com/concepts/resolution>
- [25] Olas, "Predict," Olas, 2026. Available: <https://olas.network/agent-economies/predict>
- [26] Yala, "Introduction: Yala 2.0," Yala Documentation, 2026. Available: <https://docs.yala.org/prediction-market-intelligence/introduction>
- [27] U.S. Securities and Exchange Commission, Division of Trading and Markets, "Staff Statement Regarding Broker-Dealer Registration of Certain User Interfaces Utilized to Prepare Transactions in Crypto Asset Securities," Apr. 13, 2026. Available: <https://www.sec.gov/newsroom/speeches-statements/staff-statement-regarding-broker-dealer-registration-certain-user-interfaces-utilized-prepare-staff-statement-regarding-broker-dealer-registration-certain-user-interfaces-utilized>