UKIYO NETWORK · TECHNICAL WHITE PAPER

Fulfillment Oracle

Cryptographically attested production truth on Arbitrum

Document type: Technical white paper · Version 3.0

Publication: Arbitrum One (freshness) · Ethereum L1 (finality anchor)

Classification: Public

Contents

1. Executive Summary
2. Oracle Role and Trust Boundary
3. Data Ingestion Architecture
4. Fulfillment Probability Score: Feature and Model Design
5. Attestation Scheme
6. Two-Tier Publication on Arbitrum and Ethereum
7. Cross-Chain Consumption and Finality
8. Fail-Closed State Machine
9. Multi-Source Consensus and Node Management
10. Security Analysis
11. Testing and Model Risk

1. Executive Summary

The Fulfillment Oracle is the evidence layer of Ukiyo Network. Oracle agents fuse ERP, computer vision, and logistics into a live Fulfillment Probability Score, cryptographically attested so that on-chain agents never trust a PDF or a single opaque server. The oracle ingests, scores, attests, publishes, and fails closed when feeds go dark.

The oracle publishes to Arbitrum One on every update—cheap under EIP-4844 blob economics—and anchors a Merkle root of all scores to Ethereum L1 periodically for hard finality. This two-tier design gives consumers both freshness and finality without paying L1 costs per score. This white paper specifies ingestion, scoring, attestation, publication, cross-chain consumption, fail-closed behavior, and security analysis.

2. Oracle Role and Trust Boundary

The oracle converts physical production data into a form that on-chain contracts can verify and act upon. It does not custody funds and does not execute credit actions; its authority is limited to attesting a score.

Property	Requirement	Rationale
Authenticity	Consumer can verify which oracle produced a score	Prevents impersonation
Integrity	Score cannot be altered after attestation	Prevents tampering in transit
Freshness	Consumer can reject stale scores	Prevents acting on outdated production state
Availability semantics	Absence of score is distinguishable from a score of zero	Fail-closed design requires this distinction

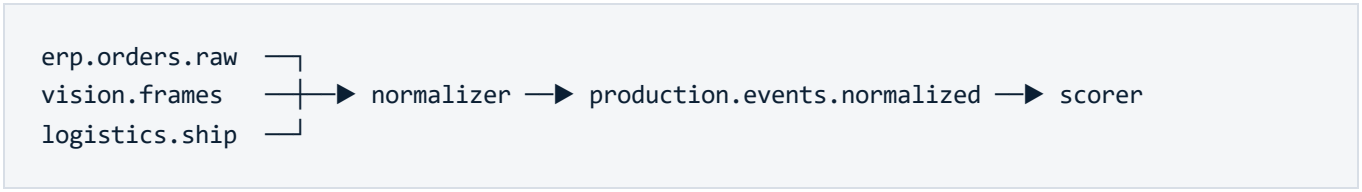
3. Data Ingestion Architecture

The oracle ingests three data classes, each with different latency, schema, and reliability characteristics.

Source	Typical latency	Schema volatility	Handling
ERP APIs	Minutes to hours	High (vendor-specific)	Connector layer with per-vendor adapters
Computer vision	Seconds	Low (image + metadata)	Streaming inference pipeline
Logistics APIs	Minutes	Medium	Polling plus webhook ingestion

3.1 Streaming backbone

Kafka is the streaming backbone. One topic per source class, schema validation at ingest, and a normalized event written to a unified topic for scoring.



The normalizer rejects events with missing required fields rather than imputing them silently. Silent imputation produces a score that appears valid but is not grounded in observed production.

4. Fulfillment Probability Score: Feature and Model Design

The Fulfillment Probability Score is a continuous estimate of on-time, to-specification fulfillment, produced by a PyTorch model. The reference feature families are standard in production-lateness prediction and are consistent with the described data sources.

Feature family	Examples	Source
Order characteristics	Notional, lead time, complexity, deadline slack	ERP

Production velocity	Units completed per shift, cycle time trend	ERP, vision
Quality signals	Defect rate, rework events, visual anomaly score	Vision
Supplier history	On-time rate, past delays, capacity utilization	ERP
Logistics state	Carrier status, transit time deviation, port congestion	Logistics

The model outputs a calibrated probability, not a raw logit. Calibration is monitored after deployment and rechecked per vertical, because a shift in Auto does not imply a shift in Pharma.

4.1 Model risk controls

- **Calibration monitoring:** compare predicted probabilities to realized outcomes in rolling windows.
- **Drift detection:** monitor feature distributions per vertical.
- **Fallback policy:** if drift exceeds a threshold, the oracle widens uncertainty or enters a degraded mode rather than publishing with false confidence.

5. Attestation Scheme

Attestation turns a score into evidence. SPOOL and downstream consumers verify EIP-712 typed data signatures that bind the score to a domain (chain ID, verifying contract) and a structured payload.

```
bytes32 constant SCORE_TYPEHASH = keccak256(
    "Score(bytes32 orderId,uint16 scoreBps,uint64 timestamp,bytes32 dataRoot)"
);

struct Score {
    bytes32 orderId;
    uint16  scoreBps;
    uint64  timestamp;
    bytes32 dataRoot; // Merkle root of supporting evidence
}
```

5.1 On-chain verification

```
function verifyScore(
    Score calldata s,
    bytes calldata signature,
    address authorizedOracle
) internal view returns (bool) {
    bytes32 digest = _hashTypedDataV4(
        keccak256(abi.encode(SCORE_TYPEHASH, s.orderId, s.scoreBps, s.timestamp, s.dataRoot))
    );
    return ECDSA.recover(digest, signature) == authorizedOracle;
}
```

The `dataRoot` field commits to supporting evidence—ERP snapshots, vision frames, logistics records—without putting it all on-chain. Consumers request the Merkle proof off-chain when needed.

Threshold signatures (k-of-n oracle nodes) are supported for scores that carry higher weight, preventing any single signer from publishing a score unilaterally.

6. Two-Tier Publication on Arbitrum and Ethereum

Attested scores are published through three channels. The choice of chain matters because attestation freshness and cost interact.

Channel	Chain	Mechanism	Trade-off
L2 registry	Arbitrum One	Contract stores latest attested score per order	Composable; cheap enough to update frequently
L1 root	Ethereum	Periodic Merkle root of scores anchored to L1	Maximum finality; batched to control cost
Event stream	Arbitrum One	Emit attested score events; consumers index	Cheapest; requires indexer
Off-chain API	—	Signed payload over HTTPS	Fastest; consumer verifies signature

6.1 Two-tier design

Scores are published to Arbitrum on every update and a Merkle root of all scores is anchored to Ethereum L1 hourly. Consumers that need L1 finality verify against the L1 root; consumers that need freshness read Arbitrum directly. The two tiers give both properties without paying L1 costs per score.

6.2 Why Arbitrum for the freshness tier

Arbitrum's ~250ms block time means a published score can be read by the next block. For SPOOL agents that pause or reprice on score changes, sub-second publication is materially better than a chain with multi-second blocks. The L1 root provides the hard guarantee; Arbitrum provides the operating cadence.

7. Cross-Chain Consumption and Finality

SPOOL is deployed on Arbitrum and consumer surfaces on Base, so the score must be readable on both. Three options are available:

- **Dual publication:** oracle signs the same score for both chains; each chain has a registry contract.
- **Canonical plus bridge:** Arbitrum is canonical; Base reads via a light-client or attestation bridge.
- **Off-chain signed payload plus on-chain verification:** consumer contracts verify the signature directly, no bridge needed.

The third option is the default because it is simplest and avoids bridge risk. On-chain registries are used only where synchronous reads are required.

7.1 Freshness under L2 finality assumptions

Arbitrum's soft confirmation is a sequencer promise, reversible until the batch reaches L1. The consumer contract checks `timestamp` against its freshness window and, for irreversible actions, waits for parent-chain safe or parent-chain final confirmation. The registry exposes `lastPublishedAt`, `state`, and `lastAnchoredRoot` so consumers can choose their finality threshold.

Consumer action	Score read from	Finality requirement
Pause advance	Arbitrum registry (soft)	None beyond sequencer inclusion
Reprice spread	Arbitrum registry (soft)	None
Advance stablecoins	Arbitrum registry, cross-checked against L1 root	Parent-chain safe

8. Fail-Closed State Machine

Fail-closed behavior is a state machine, not a single flag.

State	Trigger	Behavior
Normal	All feeds fresh; model healthy	Publish scores to Arbitrum
Degraded	One feed stale; model drift detected	Publish with uncertainty flag; consumers may widen haircuts
Paused	Critical feed absent or attestation failure	Stop publishing; emit pause event
Recovering	Feeds restored; model re-validated	Resume with conservative scores; monitor

Consumers distinguish "no score" from "score of zero." The registry exposes `lastPublishedAt` and `state` so a consumer can reject a score older than its freshness window. L1 root anchoring halts when the oracle enters Paused, so consumers relying on the root see the same fail-closed signal.

9. Multi-Source Consensus and Node Management

Independent scoring nodes are aggregated with a configurable agreement threshold.

- **Node registry:** each node registers a source name, weight, and status on-chain.
- **Weighted aggregation:** the published score is a weighted median or weighted mean of node outputs.
- **Agreement threshold:** if weighted agreement falls below a threshold, the round is void and the oracle enters Degraded or Paused.

This prevents a single compromised node from setting the score, while allowing the network to tolerate one or two faulty nodes depending on configuration.

10. Security Analysis

Threat	Mitigation	Residual risk
Source data poisoning	Multi-source fusion; anomaly detection; cross-checks	Coordinated poisoning across sources
Oracle key compromise	HSM/MPC custody; threshold signatures; key rotation	Threshold collusion
Replay across chains	EIP-712 domain separator with chainId	Misconfigured domain
Stale score acceptance	Freshness window enforced by consumer	Consumer misconfiguration
Model evasion	Adversarial testing; feature diversity	Novel production patterns
Arbitrum sequencer stall	L1 root anchoring continues independently of L2 publication	L2 consumers see stale scores until sequencer recovers

Attestation proves provenance and integrity, not accuracy. Consumers treat the score as evidence with uncertainty, not as ground truth.

11. Testing and Model Risk

- **Backtesting:** evaluate the score on historical orders with known outcomes, per vertical.

- **Shadow mode:** run the model in parallel with existing credit decisions before it drives advances.
- **Canary publication:** publish scores for a small set of orders and compare against outcomes.
- **Drift monitoring:** alert on feature distribution shift and calibration degradation.
- **Attestation testing:** Arbitrum fork tests that verify signature recovery and freshness rejection against the actual registry contract.
- **Cross-chain consistency:** verify that the L1 Merkle root matches the Arbitrum registry state at anchor time.

> **Disclaimer.** This white paper is for informational purposes and does not constitute financial, legal, or investment advice. Digital assets and DeFi protocols carry significant risk, including total loss of capital.

© 2025 Ukiyo Network. Fulfillment Oracle technical white paper, version 3.0.
Publication: Arbitrum One and Ethereum L1.