Orakai: A Decentralized Oracle Framework for AI Inference on the Blockchain

---

Abstract

Orakai introduces a decentralized oracle system optimized for AI inference integration within blockchain ecosystems. Unlike traditional data oracles, Orakai handles dynamic, non-deterministic AI-generated responses using a quorum-based response model, verifiable worker selection, and decentralized inference execution. This paper presents the full system architecture, protocol interactions, and a walkthrough of an end-to-end query using a consumer smart contract for English-to-French translation.

---

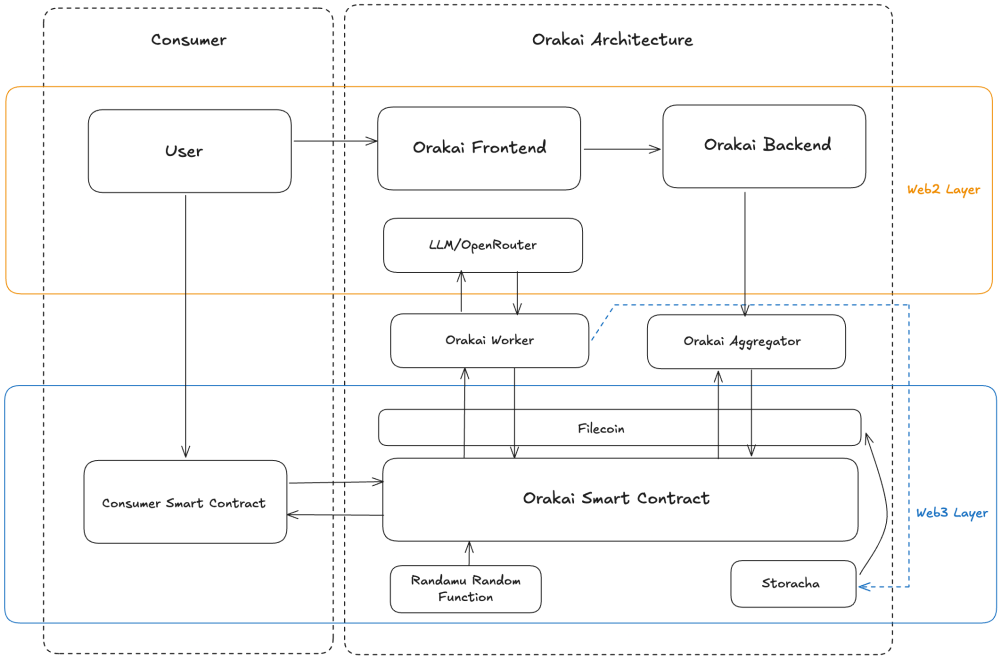
1. Introduction

Smart contracts are inherently deterministic and sandboxed, limiting their ability to interact with external, probabilistic systems like large language models (LLMs). Traditional oracles provide static data feeds (e.g., price feeds), which are inadequate for open-ended, subjective queries such as translations, sentiment classification, summarization, and more.

Orakai fills this gap by serving as a decentralized inference oracle. It allows any on-chain contract to request an LLM inference task and obtain a verifiably aggregated result from multiple independent AI agents, ensuring trust-minimized, censorship-resistant computation.

---

2. System Overview



The Orakai system consists of six primary components:

- Orakai Smart Contract (Web3 core)
- Orakai Frontend (UI layer)
- Orakai Backend (Web2 coordination layer)
- Worker Node (LLM inference executor)
- Aggregator Node (result verifier and finalizer)
- ExampleConsumerContract (example usage)

The following sections detail each of these components with a technical walkthrough.

---

3. Orakai Smart Contract

Overview

The `Orakai.sol` contract is the on-chain gateway between consumers and the decentralized AI worker network. It is written in Solidity and deployed on an EVM-compatible chain.

Responsibilities

- Accepts inference requests (`createQuery(string calldata prompt, string calldata taskType)`)
- Emits `QueryCreated` event to signal job creation
- Tracks selected worker addresses via `Randamu`
- Accepts worker-signed results (`submitWorkerResponse(bytes32 queryId, string memory resultHash, bytes signature)`)
- Waits for a threshold number (e.g. 3 of 5) of matching responses
- Calls the `Aggregator` for quorum validation
- Emits `QueryFinalized(queryId, finalResultHash)` on success

Data Structures

```
struct Query {
  address consumer;
  string prompt;
  string taskType;
  address[] assignedWorkers;
  mapping(address => string) responses;
  uint quorumCount;
}
```

Why This Matters

This contract acts as the canonical record of truth. It provides tamper-proof recording of the query lifecycle and guarantees that only majority-agreed outputs are accepted.

---

4. Orakai Frontend

Overview

The frontend is a TypeScript/React application that allows users to create and track AI queries. It interfaces with MetaMask, the Orakai smart contract, and the backend API.

Features

- Prompt input and task selection
- Wallet signature and verification
- Real-time query status updates
- Result display with provenance metadata (e.g., workers, timestamps, hashes)

Why This Matters

While Orakai is backend- and smart-contract-driven, the frontend abstracts away protocol complexity for the user, providing a clean UX to interact with decentralized inference.

---

5. Orakai Backend

Overview

Built in Go, the Orakai backend acts as an orchestrator. It does **not** perform inference or finalization itself, preserving decentralization. Instead, it handles off-chain coordination tasks like worker job distribution, storage layer writing, and reputation tracking.

Responsibilities

- Persist query metadata (timestamp, prompt, consumer address)
- Register new worker and aggregator nodes
- Route new jobs to workers via event listening
- Store all responses in decentralized storage (e.g., Storacha)
- Maintain basic usage analytics
- Expose REST APIs to frontend for query introspection

Why This Matters

This backend is necessary to bridge real-world HTTP and compute environments with the constraints of EVM smart contracts. However, it is stateless with respect to trust decisions.

---

6. Worker Node

Overview

The worker is a decentralized agent that performs inference off-chain using a configured LLM provider (e.g., OpenRouter.ai). It is a TypeScript or Go service with local inference and signing logic.

Workflow

1. Subscribes to new job notifications via backend/event indexer
2. Verifies selection via `Randamu` commitment
3. Fetches prompt and task type
4. Sends query to LLM provider (`translate("Hello", "English", "French")`)
5. Receives result (`"Bonjour"`)
6. Uploads result + metadata to `Storacha`
7. Signs result hash with its private key
8. Submits result to Orakai smart contract

Cryptographic Proofs

```
const hash = keccak256(result + queryId);
const signature = wallet.sign(hash);
```

Why This Matters

Workers provide decentralized inference, and quorum consensus ensures that adversarial or faulty workers are ignored.

---

7. Aggregator Node

Overview

The aggregator is a specialized node that monitors query responses and finalizes them based on quorum. In future versions, this will be a DAO-governed or zk-proofed process.

Responsibilities

- Watch `submitWorkerResponse` events
- Compare result hashes from workers
- Identify majority-agreed result
- Submit final result to Orakai contract via `finalizeResult(queryId, hash)`

Incentive Mechanism

- Gas refunds for aggregation
- Slashing of workers with deviating results
- Reputation score adjustment

Why This Matters

Aggregators act as light validators, finalizing responses once consensus is observed without introducing centralization risks.

---

8. ExampleConsumerContract: Language Translation

```
contract ExampleTranslator {
    address public orakaiAddress;
    mapping(bytes32 => string) public translations;

    constructor(address _orakai) {
        orakaiAddress = _orakai;
    }

    function requestTranslation(string calldata prompt) external {
        Orakai(orakaiAddress).createQuery(prompt, "translate_en_fr");
    }

    function onQueryFinalized(bytes32 queryId, string memory translatedText) external
{
        require(msg.sender == orakaiAddress);
```

```
        translations[queryId] = translatedText;
    }
}
```

---

9. Query Lifecycle Walkthrough: Translate "Hello" to French

Step-by-Step:

1. **User Input**
   User visits frontend, enters:

   ```
   Prompt: "Translate 'Hello' to French"
   Task Type: "translate_en_fr"
   ```

   → Clicks “Submit”

2. **Frontend + Backend Coordination**
   Frontend signs user intent, sends to backend → Backend stores metadata and emits job

3. **On-chain Query Registration**
   `Orakai.createQuery("Translate 'Hello' to French", "translate_en_fr")` is called

4. **Worker Selection via Randamu**
   Smart contract randomly selects 5 workers → Emits event

5. **Worker Execution**
   Each worker:

   - Queries OpenRouter.ai
   - Receives `"Bonjour"`
   - Uploads to Storacha → Gets CID
   - Signs result hash
   - Calls `submitWorkerResponse(queryId, hash, signature)`

6. **Aggregator Monitoring**
   Aggregator observes 3+ identical hashes (e.g., all workers return `"Bonjour"`)

   - Calls `finalizeResult(queryId, hash)`

7. **Smart Contract Finalization**
   Emits `QueryFinalized` with result

8. **ExampleConsumer Receives Result**
   `onQueryFinalized(queryId, "Bonjour")` is triggered on `ExampleTranslator`

9. **Storage**
   Final result is publicly available on-chain and via IPFS.

---

10. Security Assumptions

- LLM queries are subjective → quorum reduces variance
- Verifiable randomness (Randamu) prevents selection bias

- Reputation-based slashing deters bad behavior
- Aggregators are monitored → DAO governance ensures trust rotation

---

11. Future Work

- zkML proof-of-inference
- Aggregator staking + delegation model
- Query NFTs (ownership, resale)
- zk-SNARK backed worker reputation
- Multi-chain relay compatibility (Polkadot, Cosmos)
- Native OpenRouter fine-tuning

---

12. Conclusion

Orakai brings deterministic guarantees to probabilistic AI models. It bridges the gap between blockchain and intelligent inference by offering a composable, trustless, and open oracle for LLMs. With quorum-based validation, storage-backed transparency, and decentralized incentives, Orakai empowers a new class of AI-native decentralized applications.

Made by DarthBenro008