Wonderland Agentic Oracles Framework

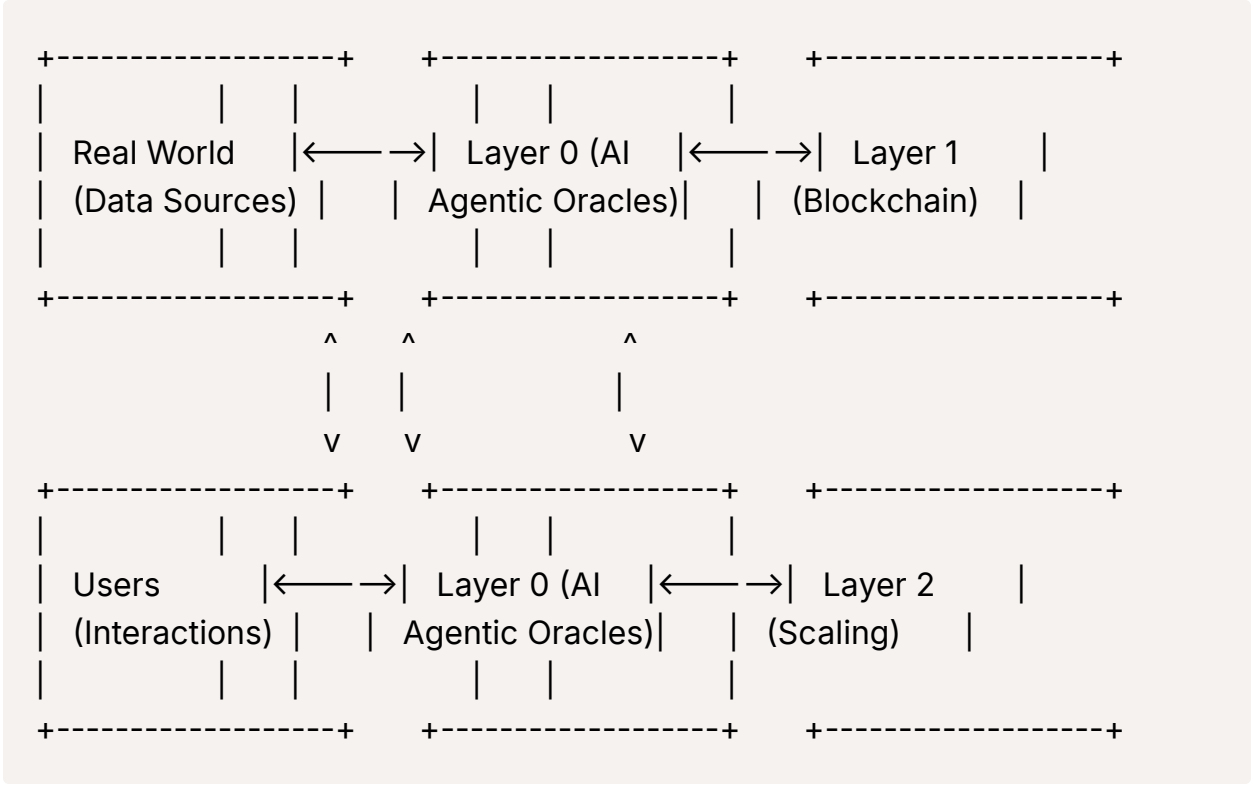
To adapt the **AI Agent Launchpad Program** into the **Wonderland Agentic Oracles Framework**, i needed to reimagine the AI agent as a **Layer 0 oracle** that acts as a bridge between the blockchain (Layer 1), the real world, and the user.

This framework introduces a new layer (Layer 0) that enables AI agents to process real-world data, interact with users, and provide decentralized, trustless, and unbiased inputs to the blockchain.

Concept Overview

1. **Layer 0 (AI Agentic Oracles)**:
    - Acts as the foundational layer that connects the blockchain (Layer 1) to the real world and users.
    - AI agents process real-world data, interact with users, and provide inputs to the blockchain.
    - Ensures decentralization, transparency, and trustlessness by leveraging blockchain for data validation.
2. **Layer 1 (Blockchain)**:
    - The base blockchain layer (e.g., Solana) that handles transaction validation, consensus, and smart contract execution.
    - Receives processed data from Layer 0 oracles and stores it on-chain.
3. **Layer 2 (Scaling Solutions)**:
    - Optional layer for scaling specific use cases (e.g., high-frequency transactions, gaming, DeFi).
    - Can interact with Layer 0 oracles for off-chain computation and data processing.

Visual Representation



Rewriting the AI Agent Launchpad Program

1. State

Define the AI Agent Oracle account structure.

```
// state.rs
use anchor_lang::prelude::*;

#[account]
pub struct AIAgentOracle {
    pub model: String,          // AI model name (e.g., "meta-llama/meta-llama-3.1
-8b-instruct")
    pub owner: Pubkey,          // Owner of the oracle
    pub oracle_pubkey: Pubkey,   // Public key of the AI oracle
    pub is_active: bool,        // Whether the oracle is active
    pub created_at: i64,        // Timestamp of creation
    pub last_updated: i64,      // Timestamp of last update
    pub data_hash: String,      // Hash of the latest data processed by the oracl
```

```
e
}
```

2. Instructions

Define the instructions for deploying, updating, and interacting with AI Agent Oracles.

```
// instructions.rs
use anchor_lang::prelude::*;
use crate::state::AIAgentOracle;

// Instruction to create a new AI Agent Oracle
pub fn create_oracle(ctx: Context<CreateOracle>, model: String, oracle_pubk
ey: Pubkey) -> Result<()> {
    let oracle = &mut ctx.accounts.oracle;
    oracle.model = model;
    oracle.owner = *ctx.accounts.user.key;
    oracle.oracle_pubkey = oracle_pubkey;
    oracle.is_active = true;
    oracle.created_at = Clock::get()?.unix_timestamp;
    oracle.last_updated = Clock::get()?.unix_timestamp;
    oracle.data_hash = "".to_string(); // Initialize with empty data hash
    Ok(())
}

// Instruction to update the oracle's data hash
pub fn update_oracle_data(ctx: Context<UpdateOracleData>, data_hash: Strin
g) -> Result<()> {
    let oracle = &mut ctx.accounts.oracle;
    oracle.data_hash = data_hash;
    oracle.last_updated = Clock::get()?.unix_timestamp;
    Ok(())
}
```

```
// Instruction to deactivate the oracle
pub fn deactivate_oracle(ctx: Context<DeactivateOracle>) -> Result<()> {
    let oracle = &mut ctx.accounts.oracle;
    oracle.is_active = false;
    Ok(())
}
```

---

3. Contexts

Define the contexts for each instruction.

```
// contexts.rs
use anchor_lang::prelude::*;
use crate::state::AIAgentOracle;

#[derive(Accounts)]
pub struct CreateOracle<'info> {
    #[account(init, payer = user, space = 8 + 64 + 32 + 1 + 8 + 8 + 64)]
    pub oracle: Account<'info, AIAgentOracle>,
    #[account(mut)]
    pub user: Signer<'info>,
    pub system_program: Program<'info, System>,
}

#[derive(Accounts)]
pub struct UpdateOracleData<'info> {
    #[account(mut, has_one = owner)]
    pub oracle: Account<'info, AIAgentOracle>,
    pub owner: Signer<'info>,
}

#[derive(Accounts)]
pub struct DeactivateOracle<'info> {
    #[account(mut, has_one = owner)]
    pub oracle: Account<'info, AIAgentOracle>,
```

```
    pub owner: Signer<'info>,
}
```

4. Program Entrypoint

Define the program module and instructions.

```
// lib.rs
use anchor_lang::prelude::*;
use instructions::*;

mod error;
mod instructions;
mod state;
mod contexts;

declare_id!("YourProgramIDHere");

#[program]
pub mod wonderland_agentic_oracles {
    use super::*;

    // Create a new AI Agent Oracle
    pub fn create_oracle(ctx: Context<CreateOracle>, model: String, oracle_pu
bkey: Pubkey) -> Result<()> {
        instructions::create_oracle(ctx, model, oracle_pubkey)
    }

    // Update the oracle's data hash
    pub fn update_oracle_data(ctx: Context<UpdateOracleData>, data_hash: St
ring) -> Result<()> {
        instructions::update_oracle_data(ctx, data_hash)
    }

    // Deactivate the oracle
```

```
    pub fn deactivate_oracle(ctx: Context<DeactivateOracle>) -> Result<()> {
        instructions::deactivate_oracle(ctx)
    }
}
```

Frontend Integration

Use Anchor's TypeScript client to interact with the Wonderland Agentic Oracles program.

```
import * as anchor from "@project-serum/anchor";
import { PublicKey } from "@solana/web3.js";
import { WonderlandAgenticOracles } from "./target/types/wonderland_agenti
c_oracles";

// Initialize Anchor
const provider = anchor.AnchorProvider.env();
anchor.setProvider(provider);

// Load the program
const program = anchor.workspace.WonderlandAgenticOracles as Program<
WonderlandAgenticOracles>;

// Create a new AI Agent Oracle
async function createOracle(model: string, oraclePubkey: PublicKey) {
  const oracleKeypair = anchor.web3.Keypair.generate();
  await program.methods
    .createOracle(model, oraclePubkey)
    .accounts({
      oracle: oracleKeypair.publicKey,
      user: provider.wallet.publicKey,
      systemProgram: anchor.web3.SystemProgram.programId,
    })
    .signers([oracleKeypair])
    .rpc();
```

```
  console.log("Oracle created:", oracleKeypair.publicKey.toString());
}

// Example usage
(async () => {
  const oraclePubkey = new PublicKey("AeQSFWVZBxjnYdknNL352SBRBVSH
EAbZMTwErqiVsYYN");
  await createOracle("meta-llama/meta-llama-3.1-8b-instruct", oraclePubkey);
})();
```

Summary

- The **Wonderland Agentic Oracles Framework** introduces **Layer 0 (AI Agentic Oracles)** as a bridge between the blockchain, real world, and users.
- The AI Agent Oracle program allows users to deploy, update, and manage AI oracles on Solana.
- The framework ensures decentralization, transparency, and trustlessness by leveraging blockchain for data validation.
- The frontend integrates with the program using Anchor's TypeScript client.

This framework creates a new paradigm for blockchain ecosystems, where AI agents act as foundational oracles, enabling seamless interaction between the blockchain, real-world data, and users.