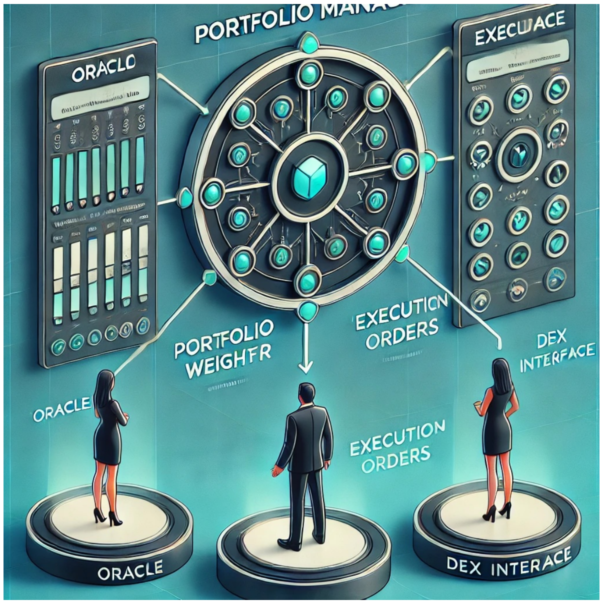
Modular Defi Crypto Portfolio System Design Document

jefferythewind



Introduction

The growing adoption of decentralized finance (DeFi) has created opportunities to automate and optimize portfolio management in the cryptocurrency space. This document presents the design of a modular and extensible system for managing crypto portfolios, leveraging the Numerai meta-model for informed decision-making. The proposed system integrates three core entities: the Oracle, Portfolio Manager, and DEX (Decentralized Exchange), each connected through well-defined interfaces.

The primary objective of this system is to facilitate efficient portfolio rebalancing, enabling users to dynamically adjust their investments based on real-time market data and predictive analytics. By adopting a modular architecture, the system ensures scalability, maintainability, and ease of integration with new platforms or data sources. This document provides an overview of the system's architecture, core components, and a Python implementation of the key modules to demonstrate its functionality.

Overview

This document outlines a modular and extensible system for managing a crypto portfolio driven by Numerai's meta-model data. The design incorporates three core entities:

- **Oracle**
- **Portfolio Manager**
- **DEX (Decentralized Exchange)**

Each entity is connected through well-defined interfaces (see Fig. 1) to ensure modularity, scalability, and maintainability.

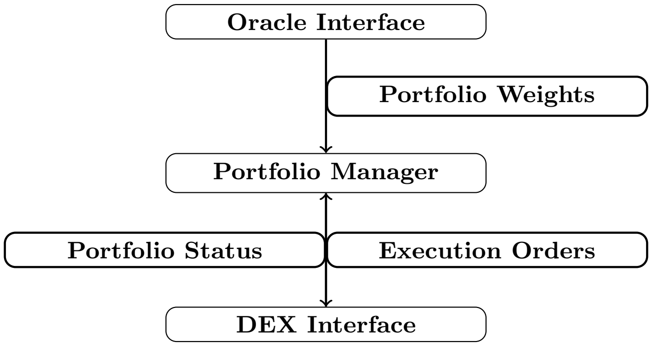


Figure 1: Portfolio Manager as Central Entity Connected to Oracle and DEX Interface.

Components

1. Oracle

Description: The Oracle computes optimal portfolio weights based on real-time data and Numerai's meta-model.

Responsibilities:

- Fetch data from external sources (e.g., real-time market data, Numerai meta-model).
- Compute and provide normalized portfolio weights.

Interface:

- `fetch_portfolio_weights(timestamp: str) -> dict`
- `validate_weights(weights: dict) -> bool`

2. Portfolio Manager

Description: The central entity that orchestrates interactions between the Oracle and DEX.

Responsibilities:

- Fetch portfolio weights from the Oracle.
- Execute portfolio rebalancing by interacting with the DEX.

Interface:

- `get_portfolio_weights() -> dict`
- `rebalance_portfolio(weights: dict) -> None`

3. DEX (Decentralized Exchange)

Description: A module to interact with decentralized exchanges for executing trades.

Responsibilities:

- Execute trades to match the portfolio weights.
- Manage USDC flows and provide portfolio performance metrics.

Interface:

- `connect(api_key: str, secret_key: str, additional_params: dict) -> None`
- `execute_trade(symbol: str, quantity: float, order_type: str) -> None`
- `rebalance_portfolio(weights: dict) -> None`

Architecture

Connections

- **Oracle → Portfolio Manager**
  - The Oracle provides normalized portfolio weights to the Portfolio Manager.

  - The Portfolio Manager requests updates from the Oracle periodically or based on triggers.

- **Portfolio Manager → DEX**
  - The Portfolio Manager sends computed weights to the DEX for execution.
  - The DEX returns execution metrics and updated portfolio status.

Data Flow

1. The Portfolio Manager pings the Oracle to fetch updated portfolio weights.
2. The Oracle computes weights using external data and validates them.
3. The Portfolio Manager receives the weights and sends them to the DEX.
4. The DEX executes trades to rebalance the portfolio, manages USDC flows, and reports back to the Portfolio Manager.

Benefits of Modular Design

- **Modularity**: Components can be developed and tested independently.
- **Scalability**: Supports the addition of new DEXs or Oracle implementations.
- **Extensibility**: Future-proofed for emerging technologies and platforms.
- **Maintainability**: Clear interfaces reduce the risk of errors when updating components.

Python Code: Portfolio Manager and Interfaces

The following code implements the Portfolio Manager class, along with generic Oracle and DEX interfaces, to demonstrate how the system can be orchestrated:

```python
from abc import ABC, abstractmethod

# Generic Oracle Interface
class OracleInterface(ABC):
    @abstractmethod
    def fetch_portfolio_weights(self, timestamp: str) -> dict:
        """Fetch portfolio weights for the given timestamp."""
        pass

    @abstractmethod
    def validate_weights(self, weights: dict) -> bool:
        """Validate the portfolio weights."""
        pass


# Generic DEX Interface
class DEXInterface(ABC):
    @abstractmethod
    def connect(self, api_key: str, secret_key: str, additional_params: dict = None):
        """Establish a connection to the DEX."""
        pass

    @abstractmethod
    def execute_trade(self, symbol: str, quantity: float, order_type: str):
        """Place a trade on the DEX."""
        pass

    @abstractmethod
    def rebalance_portfolio(self, weights: dict):
        """Rebalance the portfolio according to the provided weights."""
        pass


# Portfolio Manager Class
class PortfolioManager:
    def __init__(self, oracle: OracleInterface, dex: DEXInterface):
        """
        Initialize the Portfolio Manager with an Oracle and a DEX.

        :param oracle: An implementation of the OracleInterface.
        :param dex: An implementation of the DEXInterface.
        """
        self.oracle = oracle
        self.dex = dex

    def manage_portfolio(self, timestamp: str):
        """
        Orchestrate portfolio management by fetching weights from the Oracle
        and executing trades on the DEX.

        :param timestamp: A string representing the current time.
        """
        # Step 1: Fetch portfolio weights
        print(f"Fetching portfolio weights for timestamp: {timestamp}")
        weights = self.oracle.fetch_portfolio_weights(timestamp)

        # Step 2: Validate the weights
        if not self.oracle.validate_weights(weights):
            raise ValueError("Invalid portfolio weights received from Oracle!")

        print(f"Portfolio weights: {weights}")

        # Step 3: Rebalance portfolio on the DEX
        print("Rebalancing portfolio on the DEX...")
        self.dex.rebalance_portfolio(weights)
        print("Portfolio rebalanced successfully!")


# Example Implementation of OracleInterface
class ExampleOracle(OracleInterface):
    def fetch_portfolio_weights(self, timestamp: str) -> dict:
        # Example weights computation logic
        return {"BTC": 0.4, "ETH": 0.3, "USDC": 0.3}

    def validate_weights(self, weights: dict) -> bool:
        # Example validation: weights should sum to 1
        return abs(sum(weights.values()) - 1.0) < 1e-6


# Example Implementation of DEXInterface
class ExampleDEX(DEXInterface):
    def connect(self, api_key: str, secret_key: str, additional_params: dict = None):
        print("Connected to Example DEX.")

    def execute_trade(self, symbol: str, quantity: float, order_type: str):
        print(f"Executed {order_type} trade for {symbol} with quantity {quantity}.")

    def rebalance_portfolio(self, weights: dict):
        for symbol, weight in weights.items():
            self.execute_trade(symbol, weight, "rebalance")


# Example Usage
if __name__ == "__main__":
    oracle = ExampleOracle()
    dex = ExampleDEX()

    # Initialize Portfolio Manager
    portfolio_manager = PortfolioManager(oracle, dex)

    # Perform portfolio management
    portfolio_manager.manage_portfolio(timestamp="2024-12-13T10:00:00Z")
```

Listing 1: Portfolio Manager with Oracle and DEX Interfaces

Future Work

- Implement Portfolio Manager
- Implement first Oracle interface.
- Implement the first DEX interface.