A Multi-Volume Guide to Upgrading iris-v4 into a Hybrid AI-Web3 Product

This guide provides a detailed roadmap to transform the **iris-v4** project—a modular, multi-agent AI system—into a hybrid AI-Web3 product. By integrating blockchain functionality, we enable agents to interact with Ethereum and other Web3 ecosystems, unlocking use cases like DeFi automation, governance, and tokenized services. The guide leverages the project's existing architecture, as described in the provided context, and incorporates Web3 primitives for scalability and monetization.

Volume I: Connecting to the Blockchain

Chapter 1: The Vision of Hybrid Intelligence

The **iris-v4** project, built with **FastAPI**, **LangGraph**, and a **Flutter frontend**, is a robust platform for managing AI agents. To make it a hybrid AI-Web3 product, we aim to:

- Enable agents to query on-chain data (e.g., wallet balances, transaction history).
- Allow agents to prepare transactions (e.g., token swaps, DAO votes) for user approval.
- Support use cases like:
  - **DeFi Automation**: Agents that monitor DEX prices and execute trades.
  - **Governance**: Agents that vote in DAOs based on user-defined rules.
  - **Token-Gated Services**: Premium AI features accessible via NFTs or tokens.

This vision aligns with the project's modular architecture, where **agent_api** manages agents, **MCP servers** provide tools, and the **frontend** enables user interaction.

Chapter 2: Building the web3-mcp

Objective: Create a new **web3-mcp** service to handle blockchain interactions, integrating with **agent_api**'s tool registry.

Implementation:

- **Directory Structure**: Add `mcp-servers/web3-mcp/` to the project.
- **Dependencies** (in `requirements.txt`):

```
fastapi
uvicorn
web3
python-dotenv
```

- **Tools**:
  - `get_eth_balance(address: str)`: Queries the ETH balance of an address.
  - `get_transaction_receipt(tx_hash: str)`: Retrieves transaction details.
  - `prepare_transaction(to: str, value: int, data: str, from_address: str)`: Prepares an unsigned transaction.

- **Code Example** (in `web3-mcp/server.py`):

```
from fastapi import FastAPI
from web3 import Web3
from pydantic import BaseModel
import os

app = FastAPI()
w3 = Web3(Web3.HTTPProvider(os.getenv("ETHEREUM_RPC_URL")))

class GetBalanceRequest(BaseModel):
    address: str

@app.post("/get_eth_balance")
def get_eth_balance(request: GetBalanceRequest):
    balance = w3.eth.get_balance(request.address)
    return {"balance": balance}

class PrepareTransactionRequest(BaseModel):
    to: str
    value: int
    data: str = ""
    from_address: str

@app.post("/prepare_transaction")
def prepare_transaction(request: PrepareTransactionRequest):
    tx = {
        "to": request.to,
        "value": request.value,
        "data": request.data,
        "gas": 21000,
        "gasPrice": w3.eth.gas_price,
        "nonce": w3.eth.get_transaction_count(request.from_address),
    }
    return {"unsigned_transaction": tx}
```

- **Integration**:
  - Register tools with **FastMCP** for discovery by **agent_api**.
  - Update **agent_api/core/tool_registry.py** to include **web3-mcp** tools.
- **Environment Variables**:
  - `ETHEREUM_RPC_URL`: URL for Ethereum node (e.g., Infura, Alchemy).

Table: web3-mcp Tools

Tool Name	Description	Input Parameters	Output
get_eth_balance	Get ETH balance of an address	address: str	balance: int
get_transaction_receipt	Fetch transaction details	tx_hash: str	receipt: dict

Tool Name	Description	Input Parameters	Output
prepare_transaction	Prepare unsigned transaction	to: str, value: int, data: str, from_address: str	unsigned_transaction: dict

Chapter 3: Empowering Agents with On-Chain Actions

Objective: Enable agents to perform on-chain actions while ensuring user control over transaction signing.

Implementation:

- **Agent Logic**:
  - Modify **agent_api/core/chat_manager.py** to store user wallet addresses in chat contexts.
  - Example: When a chat starts, the frontend sends `{"type": "start_chat", "user_wallet": "0x..."}`.
  - Agents use **web3-mcp** tools to prepare transactions when needed (e.g., sending ETH).
- **Frontend Integration**:
  - Add **Web3Auth Flutter SDK** to `frontend/cyrene_ui/pubspec.yaml`:

```
dependencies:
  web3auth_flutter: ^2.0.0
```

  - Create a new service (`frontend/cyrene_ui/lib/services/web3_service.dart`) to handle wallet connections:

```dart
import 'package:web3auth_flutter/web3auth_flutter.dart';

class Web3Service {
  Future<String> connectWallet() async {
    await Web3AuthFlutter.init(Web3AuthOptions(
      clientId: 'YOUR_WEB3AUTH_CLIENT_ID',
      network: Network.mainnet,
    ));
    final userInfo = await Web3AuthFlutter.login(LoginParams(
      loginProvider: Provider.google,
    ));
    return userInfo.privKey; // Use for signing
  }

  Future<String> signTransaction(Map<String, dynamic> tx) async {
    // Sign transaction using Web3Auth
    // Implementation depends on Web3Auth SDK
  }
}
```

  - Update **ws_api** to send unsigned transactions to the frontend via WebSocket.
- **Flow**:
  - Agent detects need for action (e.g., "Send 0.1 ETH to 0x...").

- Calls `prepare_transaction` from **web3-mcp**.
- **agent_api** sends unsigned transaction to frontend via **ws_api**.
- Frontend prompts user to sign using Web3Auth.
- Signed transaction is broadcast to the blockchain.

Security:

- Users control transaction signing via their wallets, ensuring no private keys are stored on the server.

Volume II: Advanced Features

Chapter 4: NFT-Based Memory and Identity

Objective: Allow agents to mint NFTs as verifiable records of milestones or outputs.

Implementation:

- **Smart Contract**:
  - Deploy an ERC-721 contract (`AgentMemoryNFT`) on Ethereum or Polygon.
  - Example (in `smart-contracts/AgentMemoryNFT.sol`):

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

contract AgentMemoryNFT is ERC721 {
    uint256 public tokenIdCounter;

    constructor() ERC721("AgentMemory", "AM") {
        tokenIdCounter = 0;
    }

    function mint(address to, string memory tokenURI) public returns
(uint256) {
        uint256 newTokenId = tokenIdCounter;
        _safeMint(to, newTokenId);
        _setTokenURI(newTokenId, tokenURI);
        tokenIdCounter++;
        return newTokenId;
    }
}
```

- **Metadata Storage**:
  - Use **NFT.Storage** to upload metadata to IPFS.
  - Example metadata:

```
{
  "name": "Agent Memory #1",
  "description": "A milestone from Agent X",
```

```
  "attributes": [
    {"trait_type": "Timestamp", "value": "2025-08-09T01:20:00Z"},
    {"trait_type": "Action", "value": "Insightful Response"}
  ]
}
```

- **web3-mcp Tool**:
  - Add `mint_memory_nft(metadata: dict, user_wallet: str)` to `web3-mcp/server.py`:

```
import json
from nft_storage import NFTStorageClient

client = NFTStorageClient(os.getenv("NFT_STORAGE_API_KEY"))

@app.post("/mint_memory_nft")
def mint_memory_nft(metadata: dict, user_wallet: str):
    metadata_json = json.dumps(metadata)
    cid = client.store(metadata_json)
    token_uri = f"ipfs://{cid}"
    contract_address = os.getenv("NFT_CONTRACT_ADDRESS")
    abi = [...] # ABI of AgentMemoryNFT
    contract = w3.eth.contract(address=contract_address, abi=abi)
    tx = contract.functions.mint(user_wallet,
token_uri).buildTransaction({
        "from": user_wallet,
        "gas": 200000,
        "gasPrice": w3.eth.gas_price,
        "nonce": w3.eth.get_transaction_count(user_wallet),
    })
    return {"unsigned_transaction": tx}
```

- **Agent Logic**:
  - Update **agent_api**'s LangGraph workflow to trigger NFT minting on milestones (e.g., after a significant task).
  - Example: Agent checks if a response meets criteria (e.g., high user rating) and calls `mint_memory_nft`.
- **Frontend**:
  - Add a screen (`frontend/cyrene_ui/lib/screens/nft_gallery.dart`) to display minted NFTs.

Chapter 5: Decentralized AI Inference and Compute

Objective: Offload compute-intensive tasks to decentralized GPU networks.

Implementation:

- **Platform**: Use Nosana (Solana) or NEAR's AI infrastructure.
- **New Component**: Create `mcp-servers/decentralized-compute-mcp/`.
- **Tools**:

    - `run_inference(model_id: str, data: dict)`: Offload inference tasks.
- **Challenges**:
    - Requires significant changes to **agent_api** for task delegation.
    - Consider for future development due to complexity.

Volume III: The Business Model

Chapter 6: Tokenized Incentives and Payments

Objective: Implement on-chain payments for agent services.

Implementation:

- **Smart Contract** (in `smart-contracts/PaymentContract.sol`):

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract PaymentContract {
    address public owner;
    uint256 public feePerQuery;

    constructor(uint256 _feePerQuery) {
        owner = msg.sender;
        feePerQuery = _feePerQuery;
    }

    function payForQuery() public payable {
        require(msg.value >= feePerQuery, "Insufficient payment");
    }
}
```

- **payments-mcp**:
    - Create `mcp-servers/payments-mcp/` with tools like `deduct_payment(user_wallet: str, amount: int)`.
- **Integration**:
    - **agent_api** tracks usage and triggers payments.
    - Frontend displays token balances and prompts for approvals.

Chapter 7: Launching the Ecosystem - A Hybrid Launchpad

Objective: Create a platform for launching new agents and tokens.

Implementation:

- Extend **frontend/cyrene_ui** to allow users to configure and deploy agents.
- Integrate with existing launchpads (e.g., ChainGPT Pad) or build a custom one.
- Use a DAO for governance, allowing token holders to vote on new features.

Upgraded Directory Tree

```
taufeeqai-iris-v4/
├── agent/
│   ├── agent_api/
│   │   ├── __init__.py
│   │   ├── api/
│   │   │   └── main.py
│   │   ├── core/
│   │   │   ├── agent_manager.py
│   │   │   ├── chat_manager.py
│   │   │   └── ...
│   │   ├── models/
│   │   │   └── ...
│   │   ├── routes/
│   │   │   ├── agents.py
│   │   │   ├── chat.py
│   │   │   └── ...
│   │   └── ...
│   ├── db_core/
│   │   ├── alembic/
│   │   │   └── ...
│   │   ├── models/
│   │   │   └── ...
│   │   └── ...
│   ├── ws_api/
│   │   ├── __init__.py
│   │   ├── models/
│   │   │   └── ...
│   │   ├── routers/
│   │   │   ├── chat_stream.py
│   │   │   └── ...
│   │   └── ...
│   └── ...
├── frontend/
│   ├── cyrene_ui/
│   │   ├── android/
│   │   │   └── ...
│   │   ├── ios/
│   │   │   └── ...
│   │   ├── lib/
│   │   │   ├── main.dart
│   │   │   ├── models/
│   │   │   ├── screens/
│   │   │   │   ├── chat_screen.dart
│   │   │   │   ├── nft_gallery.dart
│   │   │   │   └── ...
│   │   │   ├── services/
│   │   │   │   ├── api_service.dart
│   │   │   │   ├── auth_service.dart
│   │   │   │   ├── web3_service.dart
│   │   │   │   └── ...
│   │   │   └── widgets/
│   │   │       └── ...
```

```
│   │   ├── pubspec.yaml
│   │   └── ...
│   └── ...
├── mcp-servers/
│   ├── web-mcp/
│   │   ├── Dockerfile
│   │   ├── requirements.txt
│   │   ├── server.py
│   │   └── ...
│   ├── finance-mcp/
│   │   ├── Dockerfile
│   │   ├── requirements.txt
│   │   ├── server.py
│   │   └── ...
│   ├── rag-mcp/
│   │   ├── Dockerfile
│   │   ├── requirements.txt
│   │   ├── server.py
│   │   └── ...
│   ├── telegram-mcp/
│   │   ├── Dockerfile
│   │   ├── requirements.txt
│   │   ├── server.py
│   │   └── ...
│   ├── discord-mcp/
│   │   ├── Dockerfile
│   │   ├── requirements.txt
│   │   ├── server.py
│   │   └── ...
│   ├── web3-mcp/
│   │   ├── Dockerfile
│   │   ├── requirements.txt
│   │   ├── server.py
│   │   └── ...
│   ├── payments-mcp/
│   │   ├── Dockerfile
│   │   ├── requirements.txt
│   │   ├── server.py
│   │   └── ...
│   └── ...
├── smart-contracts/
│   ├── AgentMemoryNFT.sol
│   ├── PaymentContract.sol
│   └── ...
├── docker-compose.yml
├── README.md
└── ...
```

Conclusion

This guide provides a detailed plan to upgrade **iris-v4** into a hybrid AI-Web3 product, leveraging its existing architecture and adding Web3 capabilities. The **web3-mcp** and **payments-mcp** integrate seamlessly with the

MCP framework, while **Web3Auth** ensures secure wallet interactions in the frontend. Future enhancements like decentralized compute and a launchpad can further expand the ecosystem.

Citations:

- Web3Auth Flutter SDK
- web3.py Documentation
- OpenZeppelin ERC-721
- NFT.Storage API