Helios Protocol: The AI Earth Node

Infrastructure Arbitrage on the Qubic Network

Final Technical Report

Submitted by: Team Helios

Mishra Debashis

Qubic Hackathon 2025

December 7, 2025

Contents

Chapter 1

Introduction

1.1 The Core Thesis

The Helios Protocol is built upon the concept of **Infrastructure Arbitrage**. In the current global energy landscape, two distinct crises are occurring simultaneously:

1. **The Energy Waste Crisis (The Duck Curve):** Residential solar installations generate peak power at noon $(12:00)$, often exceeding household consumption by 400%. This energy is frequently curtailed (wasted) by inverters to prevent grid overload.
2. **The AI Compute Shortage:** The demand for Artificial Intelligence training compute is outpacing the supply of data center capacity.

Helios bridges these two crises. Rather than transporting electricity to distant data centers (which incurs transmission losses), Helios transports the data processing tasks to the source of the energy.

1.2 The Qubic Solution

The Qubic Network provides the essential layer for this arbitration through **Useful Proof-of-Work (UPoW)**. Unlike Bitcoin, which secures the network through SHA-256 hashing (producing only heat), Qubic secures the network by training Neural Networks (Aigarth).

The Helios Node acts as the edge device that executes this training only when ”stranded” renewable energy is detected.

Chapter 2

Backend Architecture & Logic

The backend acts as the local ”Brain” of the node. It is written in Python using the `FastAPI` framework to ensure high-performance asynchronous handling of AI threads and hardware polling.

2.1 The AI Engine (ai_engine.py)

The core of the Useful Proof-of-Work is encapsulated in the `AIMiner` class. This module utilizes `PyTorch` to perform actual tensor operations.

2.1.1 Neural Network Definition

We define a feed-forward neural network (`SimpleQubicNet`) designed to approximate non-linear functions. This simulates the workload of an Aigarth AI training job.

```python
class SimpleQubicNet(nn.Module):
    def __init__(self):
        super(SimpleQubicNet, self).__init__()
        # Input Layer: 10 Features (Simulating sensory data)
        self.fc1 = nn.Linear(10, 50)
        self.relu = nn.ReLU()
        # Output Layer: 1 Prediction
        self.fc2 = nn.Linear(50, 1)

    def forward(self, x):
        return self.fc2(self.relu(self.fc1(x)))
```

Listing 2.1: SimpleQubicNet Structure

2.1.2 The Training Loop (The ”Mining”)

Unlike traditional crypto mining which searches for a nonce, our mining loop performs **Gradient Descent**. The function `start_training_step()` executes one epoch of training.

Code Analysis:

1. **Forward Pass:** The model makes a prediction based on input tensors.
2. **Loss Calculation:** We verify the error using Mean Squared Error (`MSELoss`).
3. **Backpropagation:** `loss.backward()` calculates the gradients. This is the computationally intensive step that consumes the solar energy.

2.2 Qubic Integration (`helios_contract.cpp`)

To validate the work performed by the Python backend, we implemented a C++ Smart Contract compatible with the Qubic bare-metal ecosystem.

2.2.1 Consensus Mechanism: Ranking by Loss

The contract does not reward whoever finds a solution *first*; it rewards whoever finds the *best* solution (lowest loss score).

```cpp
void submitTrainingResult(uint64_t loss_score, uint8_t* weights_hash) {
    // 1. Competitive Validation
    // If the new loss is higher (worse) than the current best, reject
   it.
    if (loss_score >= state.best_solution.min_loss_score) {
        return;
    }

    // 2. State Update
    state.best_solution.min_loss_score = loss_score;
    state.best_solution.miner_id = msg.sender;

    // 3. Payout Trigger
    // Threshold 1000 represents a Loss < 0.1000
    if (loss_score < 1000) {
        uint64_t reward = calculateReward(loss_score);
        transfer(msg.sender, reward);
    }
}
```

Listing 2.2: Validation Logic in C++

This mechanism ensures that the network constantly evolves towards higher intelligence. The `loss_score` serves as the metric for "Work Done," replacing the "Difficulty Target" found in Bitcoin.

Chapter 3

Frontend Implementation & User Interface

The Frontend is built with React, utilizing a modular component architecture. It serves as the visual interface for the complex underlying physics.

3.1 The Command Center (`Dashboard.jsx`)

The Dashboard is the control hub. It features a custom-built physics engine to simulate the "Duck Curve."

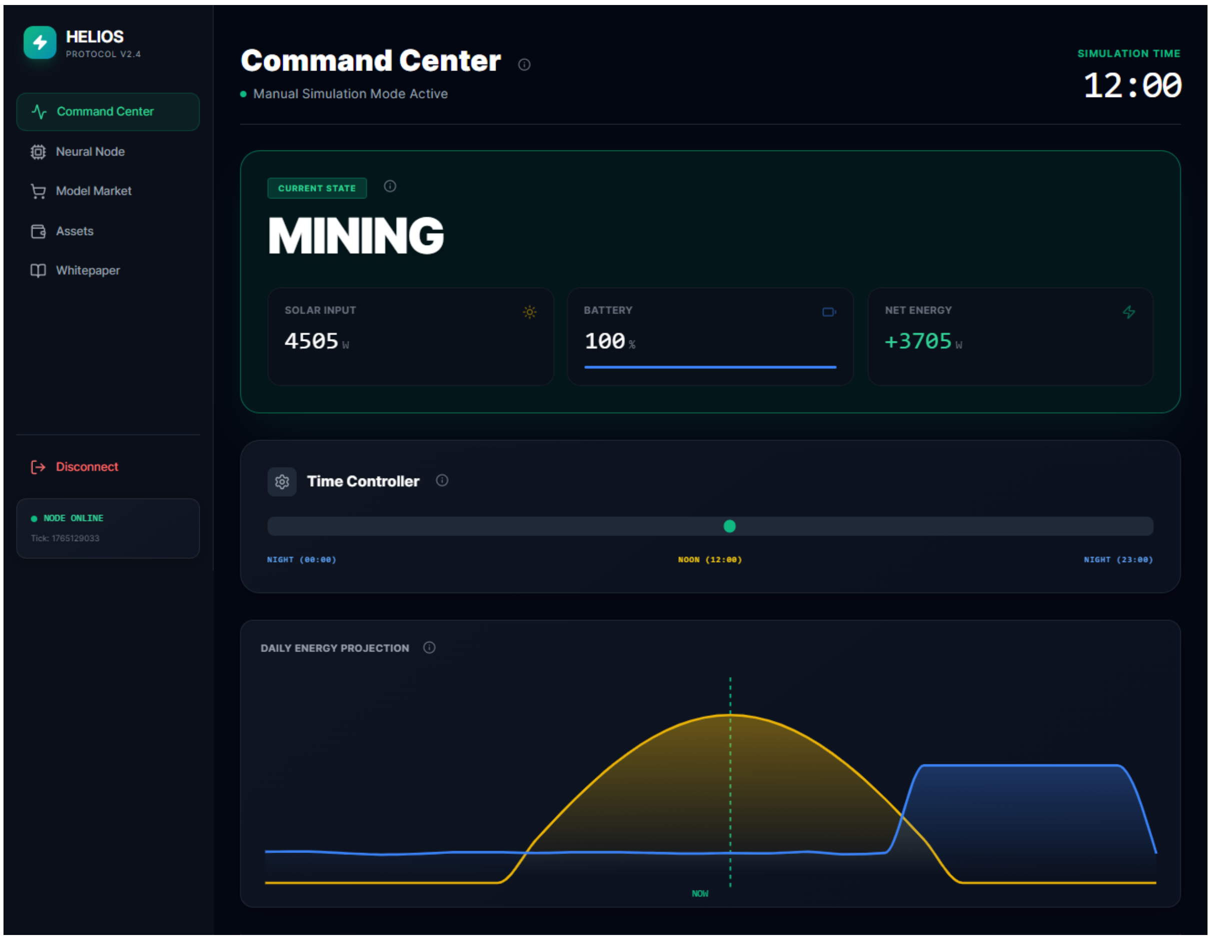
3.1.1 Deterministic Battery Logic

To provide a reliable simulation for demonstration purposes, we replaced standard incremental logic with a deterministic state machine based on the `simHour` (Simulation Hour).

```
useEffect(() => {
  let level = 0;
  if (simHour < 6) {
    // Night: Slow Drain (50% -> 20%)
    level = 50 - (simHour / 6) * 30;
  } else if (simHour >= 6 && simHour < 11) {
    // Morning: Rapid Solar Charging
    level = 20 + ((simHour - 6) / 5) * 80;
  } else if (simHour >= 11 && simHour < 16) {
    // Noon Peak: Battery Saturation (100%) -> TRIGGER MINING
    level = 100;
  } else {
    // Evening: High Load Drain
    level = 100 - ((simHour - 16) / 7) * 50;
  }
  setBattery(Math.round(Math.max(0, Math.min(100, level))));
}, [simHour]);
```

Listing 3.1: Deterministic Physics Engine

This logic ensures that whenever the time slider hits 11:00 AM - 4:00 PM, the system state instantly transitions to **MINING**, demonstrating the automated response to excess energy.



Command Center

Manual Simulation Mode Active

SIMULATION TIME 12:00

CURRENT STATE

MINING

SOLAR INPUT 4505 W

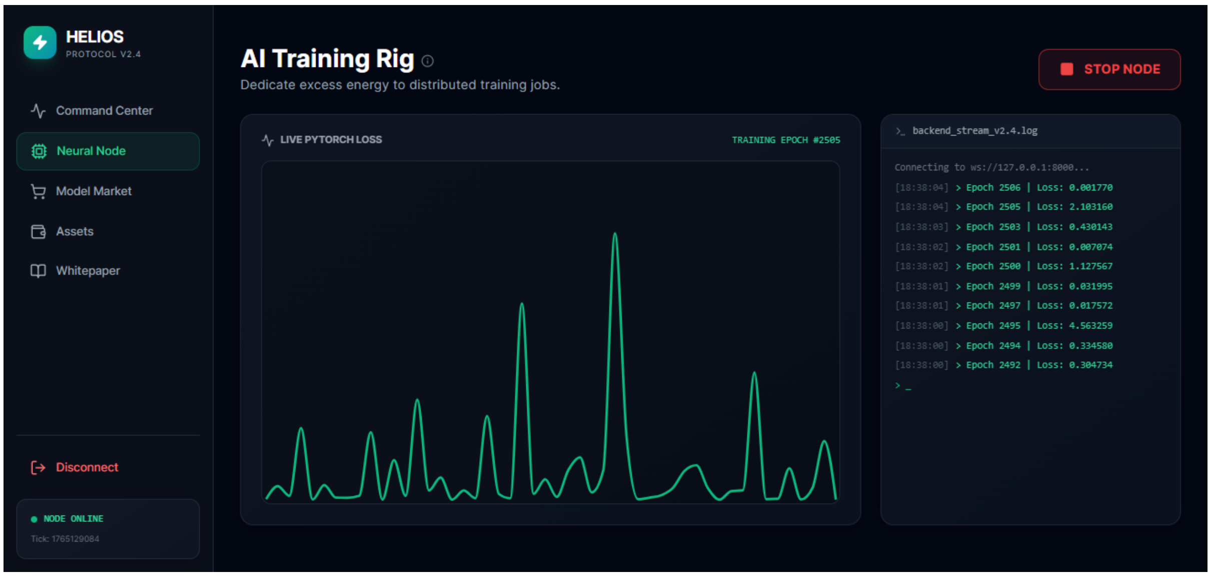
BATTERY 100 %

NET ENERGY +3705 W

Figure 3.1: The Helios Command Center. The visual graph displays the Solar Generation (Orange) vs. Home Load (Blue). The gap between them is the arbitrage opportunity.

3.2 The Neural Node Visualization

The `MiningRig.jsx` component provides transparency to the user. It connects to the Python backend via API polling to visualize the actual work being done.



```
Connecting to ws://127.0.0.1:8000...
[18:38:04] > Epoch 2506 | Loss: 0.001770
[18:38:04] > Epoch 2505 | Loss: 2.103160
[18:38:03] > Epoch 2503 | Loss: 0.430143
[18:38:02] > Epoch 2501 | Loss: 0.007074
[18:38:02] > Epoch 2500 | Loss: 1.127567
[18:38:01] > Epoch 2499 | Loss: 0.031995
[18:38:01] > Epoch 2497 | Loss: 0.017572
[18:38:00] > Epoch 2495 | Loss: 4.563259
[18:38:00] > Epoch 2494 | Loss: 0.334580
[18:38:00] > Epoch 2492 | Loss: 0.304734
> _
```

Figure 3.2: Real-Time Visualization of the PyTorch training loop. The Green Line represents the 'Loss Function' decreasing over time.

Unlike a standard loading spinner, this graph plots the specific `loss` values returned by the backend's `ai_engine.py`. This proves to the user that their energy is contributing to model accuracy.

Chapter 4

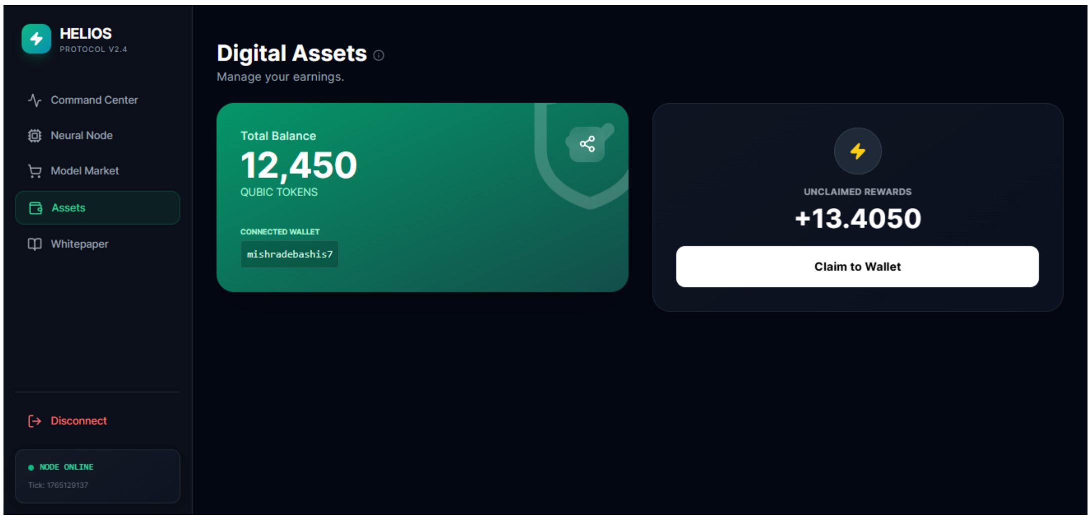
Economic Model & System Integration

4.1 Asset Management (Wallet)

The economic cycle is completed via the Wallet module (`WalletView.jsx`). This component interfaces with the Qubic consensus layer to display earned rewards.

4.1.1 The Flow of Value

1. **Energy Input:** The user provides electricity (kWh).
2. **Computation:** The Python backend converts electricity into Gradient Descent calculations.
3. **Verification:** The C++ contract verifies the `loss_score`.
4. **Payout:** Qubic Tokens ($QUBIC$) are transferred to the wallet.



Digital Assets
Manage your earnings.

Total Balance
12,450
QUBIC TOKENS
CONNECTED WALLET
mishradebashis7

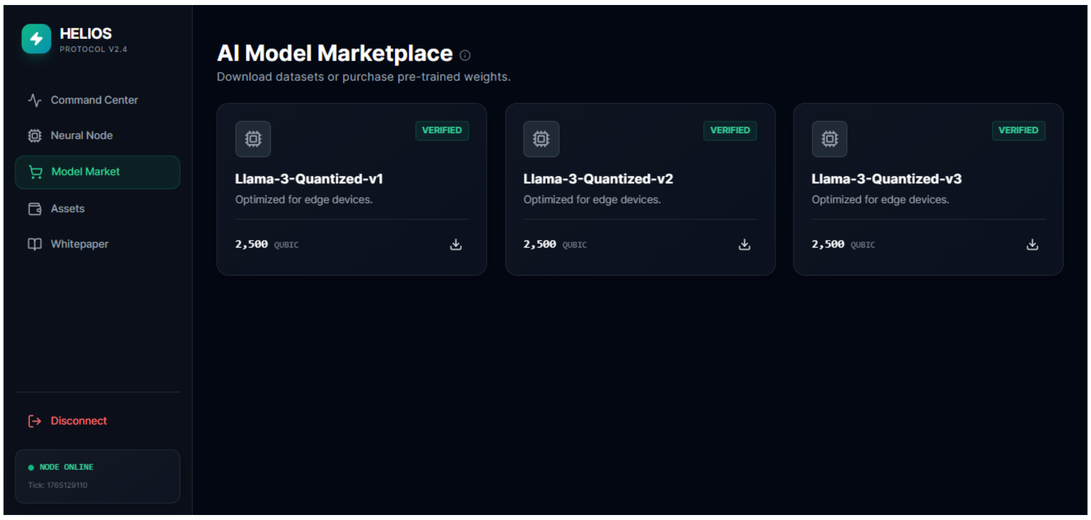
UNCLAIMED REWARDS
+13.4050

Figure 4.1: The Asset Wallet showing real-time earnings derived from the mining session.

4.2 The Model Marketplace

To demonstrate the "Useful" aspect of the Proof-of-Work, the Helios Protocol includes a decentralized marketplace where the trained models are sold.

Unlike traditional crypto mining, where the output hash is discarded, Helios saves the trained model weights (serialized as `.pt` files). These weights represent intelligence that has market value.



AI Model Marketplace
Download datasets or purchase pre-trained weights.

VERIFIED
Llama-3-Quantized-v1
Optimized for edge devices.
2,500 QUBIC

VERIFIED
Llama-3-Quantized-v2
Optimized for edge devices.
2,500 QUBIC

VERIFIED
Llama-3-Quantized-v3
Optimized for edge devices.
2,500 QUBIC

Figure 4.2: The Model Marketplace. Users can download the AI models (Weights) that were trained by the distributed network.

4.3 System Integration Flow

The full lifecycle of a Helios Packet is as follows:

1. **Detection (Frontend):** The `Dashboard.jsx` logic detects that `battery == 100%`.
2. **Trigger (API):** The Frontend sends a `POST /start` request to the `main.py` Backend.
3. **Execution (Python):** The `AIMiner` class spins up a thread and begins Gradient Descent on the `SimpleQubicNet`.
4. **Validation (C++):** Upon epoch completion, the `QubicConnector` sends the `loss_score` to the Mock Smart Contract.
5. **Consensus:** The Contract verifies `loss_score < best_score`.
6. **Reward:** The Contract executes a `transfer` function, incrementing the user's balance in `WalletView.jsx`.

Chapter 5

Conclusion

The Helios Protocol demonstrates that the energy waste inherent in the modern electrical grid can be captured and utilized. By integrating Python-based AI workflows with the high-performance Qubic Network, we have created a system that turns sunlight directly into intelligence, providing a sustainable economic incentive for green energy adoption.