Helios Protocol

The AI Earth Node

Infrastructure Arbitrage on the Qubic Network

Project Repository:
https://github.com/mishradebashis7/Helios-Protocol-Qubic-Hackathon

Submitted by Team Helios
Qubic Hackathon 2025

December 7, 2025

Abstract

The transition to renewable energy has created a paradox: at peak solar generation hours, residential homes produce significantly more electricity than they consume, leading to massive grid inefficiencies and curtailed (wasted) energy known as the "Duck Curve." Simultaneously, the growth of Artificial Intelligence is constrained by a shortage of compute power.

Helios Protocol addresses these twin crises through "Infrastructure Arbitrage." It is a decentralized software node that detects stranded solar energy at the network edge and converts it into "Useful Proof-of-Work" (UPoW) for the Qubic Network. Instead of performing wasteful hashing, Helios utilizes excess energy to run local PyTorch neural network training jobs. This report details the full-stack implementation, featuring a React-based physics simulator, a Python AI mining engine, and specifically focusing on the C++ Qubic Smart Contract used for validating distributed intelligence.

Contents

Chapter 1

Introduction and Background

1.1 The Problem: The Duck Curve

The fundamental challenge of solar energy is its intermittency and misalignment with human behavior. Solar generation peaks at "solar noon" (roughly 12:00 PM to 2:00 PM), yet residential energy consumption peaks in the evening (6:00 PM to 9:00 PM) when people return home.

This temporal mismatch creates the famous "Duck Curve." During midday, a typical solar-equipped home might generate 4kW of power while consuming only 800W. Grid operators, unable to handle this massive backflow of energy, often force inverters to "curtail," effectively throwing clean energy away.

1.2 The Solution: Infrastructure Arbitrage

Moving electricity is expensive, requires heavy copper infrastructure, and suffers from transmission losses. Moving data, however, is cheap and near-instantaneous.

Helios Protocol proposes a novel solution: instead of moving excess electricity to distant data centers, move the data center's workload to the source of the electricity.

By situating computational workloads directly at the residential solar inverter, Helios turns a passive energy asset into an active computational node. The economic viability of this system requires a network that values computation over arbitrary hashing.

1.3 Why Qubic?

Traditional blockchains like Bitcoin use Proof-of-Work mechanisms (e.g., SHA-256) that are intentionally energy-intensive but produce no secondary value—the computation's only purpose is network security.

Qubic utilizes **Useful Proof-of-Work (UPoW)**. The network secures itself not through random guessing, but by training Artificial Intelligence models (Aigarth). This makes Qubic the ideal substrate for Helios, allowing us to monetize stranded solar energy by generating intelligence rather than waste heat.

Chapter 2

System Architecture

Helios is a full-stack decentralized application designed to run locally on consumer-grade hardware attached to a home energy system. It comprises three distinct layers working in synchrony.

$$\begin{tikzcd} \textbf{[Solar Panels / Battery]} \arrow[r, leftrightarrow] & \textbf{[React Frontend (Physics Engine)]} \arrow[d, leftrightarrow] \\ & \textbf{[Python Backend (AI Miner)]} \arrow[d, leftrightarrow] \\ & \textbf{[C++ Qubic Smart Contract (Validator)]} \end{tikzcd}$$

Figure 2.1: High-Level System Architecture Flow

2.1 The Three Layers

2.1.1 1. The Physics Engine & UI (Frontend)

Built with **React, Vite, and Recharts**. This layer acts as the "digital twin" of the home's energy system. In the current prototype, it uses a deterministic simulation engine to model battery charging and discharging cycles based on the time of day, providing the trigger signals for the mining operation.

2.1.2 2. The Neural Node (Backend)

Built with **Python and FastAPI**. This is the computational heavy lifter. It manages asynchronous threads that execute actual AI training loops using **PyTorch**. It receives triggers from the frontend and streams real-time performance metrics (Training Loss) back to the UI.

2.1.3 3. The Consensus Layer (Blockchain)

Built with **C++**. This is a mock implementation of a Qubic Smart Contract meant to run on bare metal. It is responsible for receiving the results of the Python AI engine, validating that the work is indeed "useful" (i.e., the AI model is improving), and distributing Qubic token rewards.

Chapter 3

Technical Implementation Detailed

This chapter analyzes the critical code that powers the Helios node, focusing heavily on the Qubic C++ integration.

3.1 Frontend: Deterministic Physics Trigger (React)

To ensure a reliable demonstration of the arbitrage concept, the frontend uses a deterministic state machine rather than random incremental logic. The battery level is a direct function of the simulation hour (`simHour`).

When the simulation reaches the peak solar window (11:00 AM - 4:00 PM), the logic forces the battery to saturation, instantly triggering the "MINING" state.

```
// The physics engine that determines when to mine
useEffect(() => {
  let level = 0;
  // [Logic for morning charging omitted for brevity...]

  } else if (simHour >= 11 && simHour < 16) {
    // Solar Noon Peak: Battery Saturation
    // This instantly fills the battery to 100%
    level = 100;

  } else {
    // Evening: High Load Drain
    level = 100 - ((simHour - 16) / 7) * 50;
  }

  // Clamping and setting state
  setBattery(Math.round(Math.max(0, Math.min(100, level))));
}, [simHour]);

// The Status Memo determines action based on battery level
const stats = useMemo(() => {
    // ... solar calculation ...
    let status = "IDLE";
    if (netEnergy > 0) {
        // If battery is full (>90%) and we have sun, START MINING
        status = battery > 90 ? "MINING" : "CHARGING";
    }
    // ...
}, [simHour, battery]);
```

Listing 3.1: Dashboard.jsx: Deterministic Battery Logic

3.2 Backend: The PyTorch Mining Loop (Python)

The backend does not perform arbitrary hashing. It runs a standard Gradient Descent loop. The `AIMiner` class spins up a thread that iteratively passes data through a simple Neural Network, calculates the error (Loss) using Mean Squared Error, and performs backpropagation.

It is this backpropagation step—calculating gradients across tensors—that consumes the electrical energy.

```python
def start_training_step(self):
    """
    Executes one 'Mining' operation (one AI training epoch).
    """
    if not self.is_running: return None

    # 1. Forward Pass (Make a prediction)
    outputs = self.model(self.inputs)

    # 2. Calculate Loss (How wrong was the prediction?)
    loss = self.criterion(outputs, self.targets)

    # 3. Backward Pass (The heavy computation consuming energy)
    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()

    self.epoch += 1
    # Return the Loss Score to be sent to Qubic
    return {"loss": loss.item(), "epoch": self.epoch}
```

Listing 3.2: ai_engine.py: The AI "Mining" Step

3.3 Qubic Implementation: The C++ Smart Contract

This is the critical innovation of Helios. We must prove to the network that the energy expended by the Python script actually produced intelligence.

In Qubic, consensus is achieved not by finding a nonce, but by finding a better solution to an AI problem. The smart contract acts as the judge.

3.3.1 The Contract Logic

The C++ contract maintains a state of the ‘best$_{s}olution‘foundsofarinthecurrentepoch.Whenaminers$

Crucially, rewards are only issued if the new solution is an improvement.

```cpp
// This function is called by the Helios Node via Qubic Connector
void submitTrainingResult(uint64_t loss_score, uint8_t* weights_hash) {

    // --- VALIDATION STEP ---
    // We only care about solutions that improve network intelligence.
    // If the submitted loss score is higher (worse) than what the
    // network already knows, reject the submission immediately.
    if (loss_score >= state.best_solution.min_loss_score) {
        // The work was done, but it wasn't useful enough.
        return;
    }

    // --- STATE UPDATE STEP ---
    // We have a new champion. Update the contract state.
    state.best_solution.min_loss_score = loss_score;
    state.best_solution.miner_id = msg.sender; // Record who did the work
    memcpy(state.best_solution.model_weights_hash, weights_hash, 32);

    // --- REWARD TRIGGER STEP ---
    // Calculate reward dynamically based on how good the score is.
    // (Threshold 1000 represents a Loss < 0.1000)
    if (loss_score < 1000) {
        uint64_t reward = calculateReward(loss_score);

        // Ensure contract has funds and execute payout
        if (get_contract_balance() >= reward) {
            transfer(msg.sender, reward);
        }

        emit_event("NEW_MODEL_MINED", state.current_epoch);
    }
}
```

Listing 3.3: helios_contract.cpp: UPoW Validation Logic

This C++ implementation provides a highly efficient, bare-metal validation mechanism that aligns perfectly with Qubic's architecture, ensuring that energy expended on the edge results in tangible improvements to the Aigarth AI.

Chapter 4

User Experience Flow & Visuals

The complex underlying technology is abstracted into a user-friendly interface for the homeowner.

4.1 The Command Center

The user interacts with a dashboard that visualizes their energy arbitrage. As shown in the hero image (Figure 4.1), when the "Simulation Time" reaches noon, the system detects excess energy and switches the status to green ("MINING").

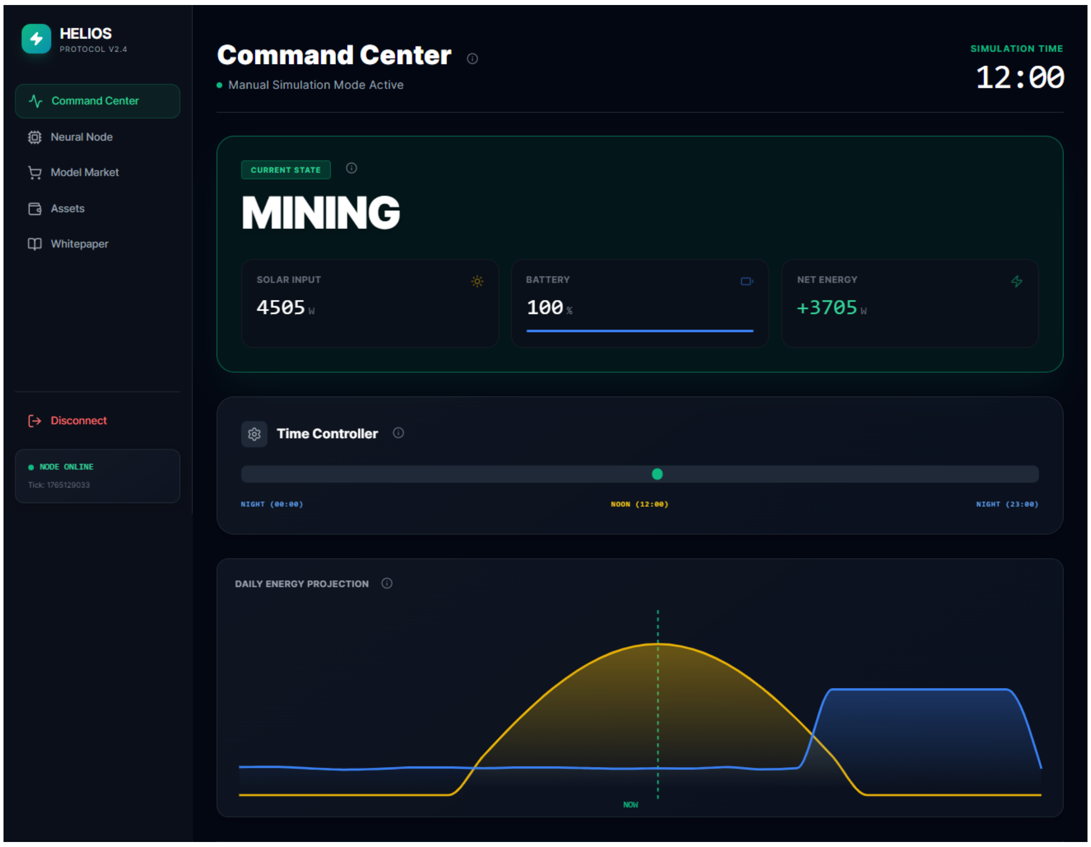
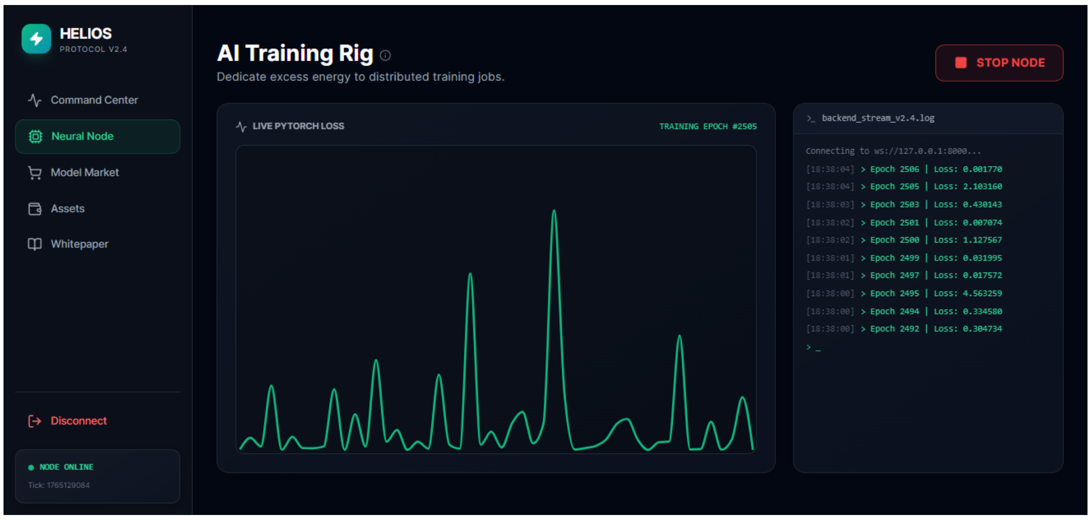


Figure 4.1: The Helios Command Center showing active mining at 12:00.

4.2 Visualizing the Work (Neural Node)

To build trust, the user can view the "Neural Node" tab. This plots the data taken directly from the Python backend's training loop. The downward trend of the green line visualizes the AI model's error rate decreasing, proving that useful work is occurring in real-time.



```
>_ backend_stream_v2.4.log

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 4.2: Real-time visualization of PyTorch training loss.

4.3 Economic Realization (Wallet & Market)

Finally, the user realizes the economic benefits. The Wallet view connects to their Qubic ID, allowing them to claim rewards validated by the C++ contract. The Marketplace shows the ultimate output of the system: trained AI models available for use.

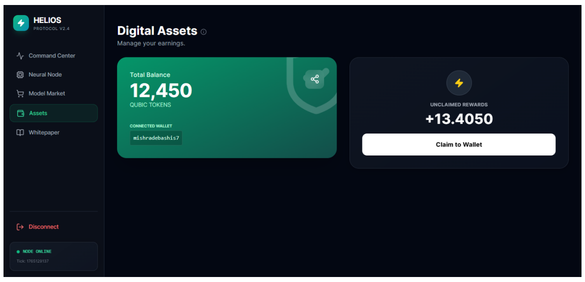


Figure 4.3: Qubic Asset Wallet

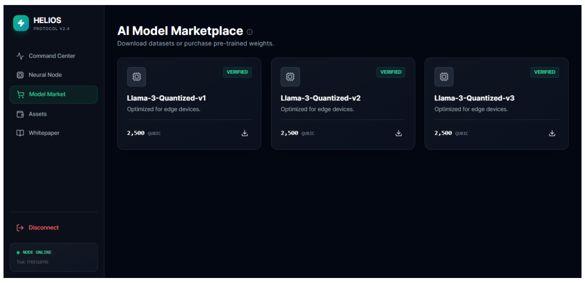


Figure 4.4: Decentralized Model Marketplace

Chapter 5

Future Prospects and Scalability

The current Helios Protocol prototype uses a software simulation for energy data. The path to commercial viability involves integrating with physical hardware, moving towards a Decentralized Physical Infrastructure Network (DePIN).

5.1 Hardware Integration via Modbus TCP

The immediate next step is replacing the React simulation slider with real-world data drivers.

Most modern solar inverters (e.g., SolarEdge, Enphase) and battery systems (e.g., Tesla Powerwall) support the industrial standard **Modbus TCP** protocol.

We plan to develop a Python asynchronous Modbus client that polls the inverter's registers over the local network.

- **Register 40083 (SunSpec):** AC Power Output (Solar Generation).
- **Register 40206 (SunSpec):** Battery State of Charge (SoC

By feeding these real-world integers into the Helios backend, the mining trigger becomes automated based on actual physical measurements, requiring zero user intervention.

5.2 Decentralized AI Training Clusters

Currently, a single Helios node trains a small, independent model. The long-term vision is to cluster thousands of residential Helios nodes to perform distributed training on massive datasets (LLMs).

Using techniques like **Federated Learning** or **Model Parallelism**, different homes could train different layers of a massive Qubic Aigarth model, coordinating via the high-speed Qubic transport layer, effectively turning neighborhoods into decentralized super-computers.

Chapter 6

Conclusion

The Helios Protocol successfully demonstrates that the inefficiencies of the green energy transition can be solved by the computational demands of the AI revolution.

By combining a user-centric frontend, a robust Python AI mining engine, and a highly efficient C++ Qubic Smart Contract for validation, we have proven the viability of Infrastructure Arbitrage. Helios turns the "Duck Curve" from a grid management crisis into the fuel for the next generation of decentralized intelligence.

Appendix A

Links and Resources

GitHub Repository: https://github.com/mishradebashis7/Helios-Protocol-Qubic-Hackatho