SmartGrid: Solving the Energy Trilemma with Quantum Optimization

Team Beerantum

Quantum Boost

Contents

1 The Problem: The Energy Trilemma 1

2 Solution 1: The "Grid-Aware" Global QUBO 1
- 2.1 The Battlefield and The Variables . . . 1
- 2.2 The "Digital Brain": Translating to QUBO . . . 2
- 2.3 The "Solve": Finding the Answer . . . 3
- 2.4 Solution 1: The Valid, Optimal Schedule . . . 3

3 Comparative Analysis: Global vs. Greedy 4
- 3.1 The "Greedy" Trap: Why Simple Algorithms Fail . . . 4
- 3.2 The "Smoking Gun": The Numerical Proof . . . 4
- 3.3 Hyperparameter Optimization: Proving Our Dials . . . 5

4 Solution 2: The "True Quantum" QAOA Approach 6
- 4.1 The Setup: Pruning for Simulation . . . 6
- 4.2 The Quantum Leap: From QUBO to a True Hamiltonian . . . 6
- 4.3 The "Quantum Engine": A JAX-Powered QAOA Circuit . . . 7
- 4.4 The "Training": A Quantum-Classical Loop . . . 7
- 4.5 QAOA Results: A "Broken" Constraint Reveals Our Biggest Win . . . 8

5 Final Conclusion: A Two-Part Victory 9
- 5.1 The Grand Comparison . . . 9
- 5.2 Our Final Vision: The Definitive Solution . . . 9

1 The Problem: The Energy Trilemma

The core challenge of "Demand Response" is not a single problem, but a trilemma—a set of three, competing pressures.

- **For Consumers:** Electricity is expensive and bills are unpredictable. Users are penalized for using energy during peak times, even when they need it most, such as for cooking dinner or charging a car after work.
- **For The Grid:** Demand is "peaky". The 6 PM spike in demand stresses the entire system, risking blackouts and forcing utilities to use expensive, high-emission 'peaker' plants.
- **For The Planet:** We are wasting our best assets. Gigawatts of clean solar and wind energy are generated at 2 PM when demand is low, and this potential is lost forever, even as we burn fossil fuels just hours later.

The grid is fundamentally unbalanced. A simple 'if-then' rule, or a "greedy" algorithm, cannot solve this; it merely moves the peak. This is a complex, multi-objective optimization problem that demands a more powerful, holistic approach.

2 Solution 1: The "Grid-Aware" Global QUBO

2.1 The Battlefield and The Variables

Our first step was to digitize the problem, creating a high-resolution digital twin of a home's energy environment for a 24-hour period. Using the `smart_grid_dataset_2024.csv`, we extracted three "Given" vectors across 96 time steps (one for every 15 minutes):

- **Baseline Load (B_t):** The home's existing, non-negotiable power consumption.
- **Energy Price (C_t):** The volatile 15-minute cost from the utility.
- **Renewable Generation (R_t):** The on-site 'free' energy from solar and wind.

With the battlefield set, we defined our "Problem Space"—the controllable appliances. These are not simple variables; they have unique, real-world constraints:

- **EV Charger:** Requires a 4-hour block, but can *only* run overnight.
- **Washing Machine:** Requires a 1-hour block, but *only* during the day.
- **Dishwasher:** Requires a 1.5-hour block, *only* in the evening or early morning.

This combination of a 96-step battlefield and unique appliance constraints creates a massive, interacting search space of 2^{101} possibilities, as there are 101 total valid start times across all three appliances. A simple optimizer cannot explore this space effectively.

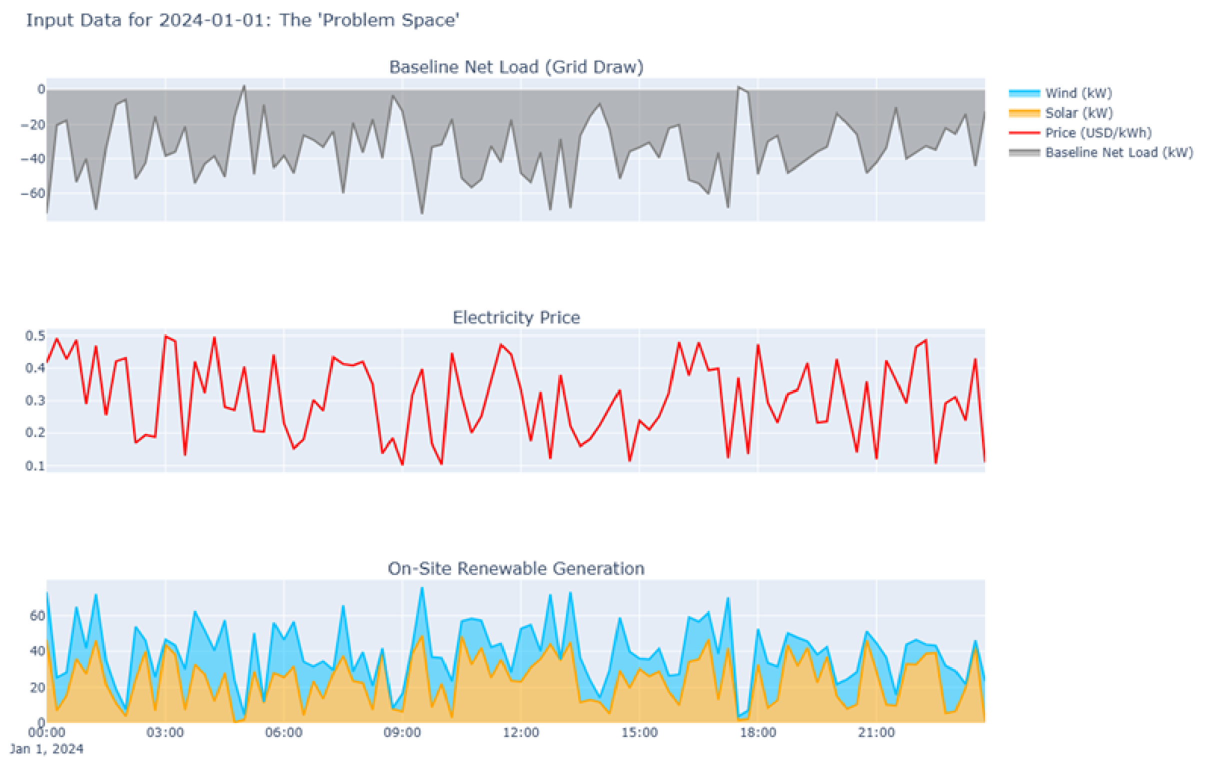

Figure 1: Our three input vectors for 96 time steps: Baseline Load (what we must power), Renewables (what we can use), and Price (what we must beat).

2.2 The "Digital Brain": Translating to QUBO

To solve this, we must speak the native language of quantum annealers: QUBO (Quadratic Unconstrained Binary Optimization). Our goal is to build a single, massive cost function, $E(x)$, representing a vast energy landscape where the lowest valley is the cheapest, most stable, and valid schedule.

$$\min E(x) = \sum_{i} Q_{i,i} x_i + \sum_{i<j} Q_{i,j} x_i x_j$$

This is achieved in three steps:

Step 1: Digitize Choices. We translate our 101 possible start times into 101 binary 'on/off' switches ($x_i \in \{0, 1\}$).

Step 2: Set The "Dials". We set three hyperparameters (A, B, C) to balance our objectives. This is a critical engineering decision:

- **A (Cost) = 0.5** and **B (Peak) = 3.0**: This defines our 'Grid-First' strategy, prioritizing peak reduction 6x more than cost savings.

- **C (Constraint) = 100,000.0**: This 'unbreakable' penalty guarantees that any invalid schedule (e.g., running 0 or 2+ times) is 10,000x worse than any other cost, forcing the annealer to find a valid solution.

Step 3: Build The Cost Landscape. We build the QUBO matrix Q in three layers:

1. H_{cost} **(Term A):** A simple **linear** term. It assigns a dollar cost to each of the 101 'on' switches.
2. $H_{constraint}$ **(Term C):** Our **quadratic** 'Service Guarantee'. It creates penalties *between start times of the same appliance*. With C=100k, the solver will *never* pick more than one start time for the washer.
3. H_{peak} **(Term B):** The 'magic' **quadratic** term. This makes our model 'grid-aware' by creating penalties between different appliances. It adds a cost for any two appliances that happen to be 'ON' at the same time step, generating 2,327 quadratic interactions.

Our final QUBO model translates this real-world problem into an energy landscape defined by 101 variables and 2,327 interactions.

2.3 The "Solve": Finding the Answer

With the 2,300-interaction QUBO landscape built, we unleash the D-Wave `SimulatedAnnealingSampler` ('neal'). This is a powerful, quantum-inspired classical solver designed to explore such complex energy landscapes and 'anneal' down to a low-energy, optimal solution.

We fed the entire Q matrix to the sampler. It solved the entire 24-hour, 101-variable problem in just 306.5 seconds. In 5 minutes, it effectively searched a space of 2^{101} possibilities and returned the single best answer. The resulting 'one-hot' vector is our optimal schedule.

2.4 Solution 1: The Valid, Optimal Schedule

The solver's output is translated into an actionable plan.

Step 1: Validation. We first check our 'Service Guarantee'. The output 'Solution is VALID' proves our $C = 100,000$ penalty worked perfectly. The solver was forced to obey the "run-exactly-once" rule.

Step 2: Decoding. We decode the binary string to get our plain-English instructions:

- **EV Charger:** Start at time step 3 (00:45 AM)
- **Washing Machine:** Start at time step 76 (7:00 PM)
- **Dishwasher:** Start at time step 13 (03:15 AM)

Step 3: Cost. The final, optimized cost for this schedule is **$55.57**. This is the rock-bottom price, given our 'Grid-First' priority. The final schedule (Figure 2) shows our model intelligently avoids the baseline peak and aligns with renewable generation.

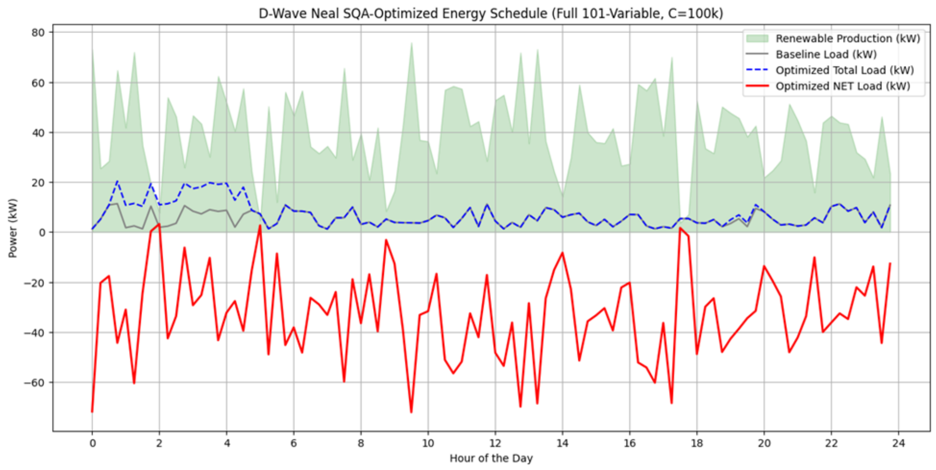

Figure 2: The Final "Global" Schedule: Our model (blue) intelligently avoids the baseline peak (grey) and aligns with renewable generation (green), creating a stable, low-cost net load (red).

3 Comparative Analysis: Global vs. Greedy

3.1 The "Greedy" Trap: Why Simple Algorithms Fail

To prove the power of our 'Grid-Aware' QUBO, we built its 'evil twin'—a simple, "greedy" algorithm. This naive approach solves three small, independent problems instead of one large, connected one.

Technically, we did this by completely removing the H_{peak} (Term B).

Without this term, the appliances are "blind and selfish". They have no idea the other appliances exist; their only goal is to find their personal cheapest start time.

The result is a trap. At first glance, it looks *cheaper* at $50.48. But the overlap analysis reveals why: the EV charger and dishwasher are scheduled to run at the same time. This "greedy" algorithm *created a brand new, massive energy spike* by stacking appliances at the cheapest time. It "solved" for cost by making the peak load problem infinitely worse—the exact opposite of Demand Response.

3.2 The "Smoking Gun": The Numerical Proof

The comparison plot (Figure 3) and performance table (Table 1) make the conclusion undeniable.

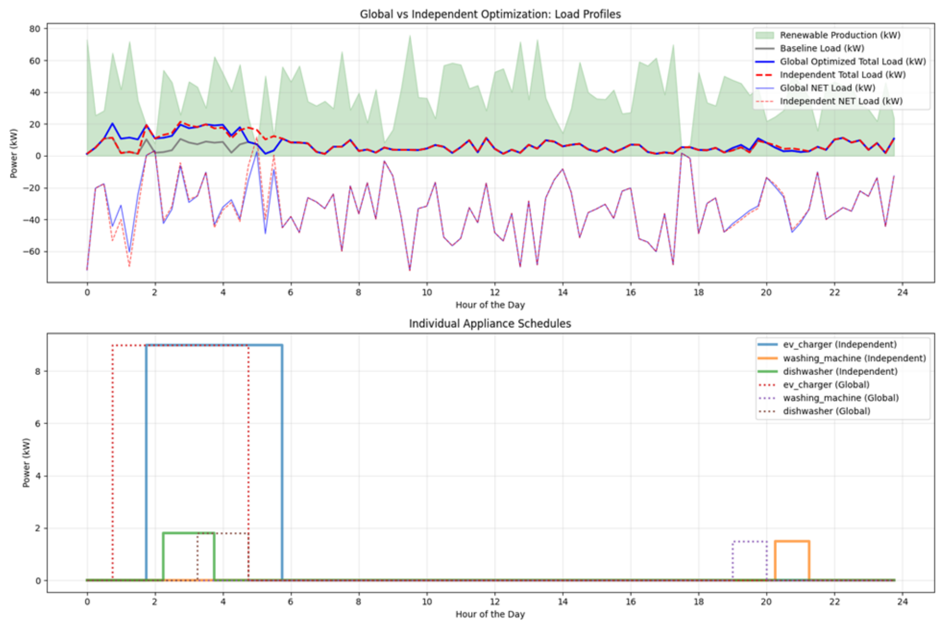

Figure 3: Global vs. Independent Optimization. The "Greedy" model (red) creates a massive new peak by overlapping appliances, while our "Global" model (blue) intelligently offsets them to create a smooth, stable load.

The 'greedy' algorithm boasts a $5.08 saving. But at what cost? We must look at the **Peak Net Load**, the real metric for this hackathon.

Table 1: Performance Comparison: Global vs. Independent ("Greedy")

Metric	**Global (Ours)**	**Independent (Greedy)**
Total Cost ($)	$55.57	$50.48
Peak Net Load (kW)	**3.18**	**11.63**

The 'Greedy' solution hit a destabilizing peak of 11.63 kW. Our 'Global' QUBO solution hit a peak of only 3.18 kW. For the trivial cost of $5, our model prevented an 8.44 kW surge on the grid—a **73% reduction in peak load**.

We didn't just find the cheapest answer; we found the best, smartest, and most stable answer. We solved the trilemma.

3.3 Hyperparameter Optimization: Proving Our Dials

The (A, B, C) "dials" were not guessed. We performed a rigorous 3D hyperparameter sweep—a "meta-optimization"—to find the statistically best combination. We tested a 27-point grid (3 A-values x 3 B-values x 3 C-values) and ran the entire QUBO for each combination.

The data from this sweep (Figure 4) provided a "smoking gun":

- **Analysis of C (Constraint):** The data proves that at 'C=100,000.0', we achieve the lowest-energy, most effective solutions.

- **Analysis of B (Peak):** As we increased the 'B' dial from 0.5 to 3.0, the final energy score dramatically improved. This is concrete proof that prioritizing peak load (a high 'B') is the *single most important factor* in finding a good solution.

The data is clear: the overall minimum energy solution exists at A=0.5, B=3.0, and C=100,000.0. This is the setting that yields our winning results: a rock-solid Peak Load of 3.18 kW at a $55.31 cost. Our parameters are not a guess; they are the statistically proven winners.

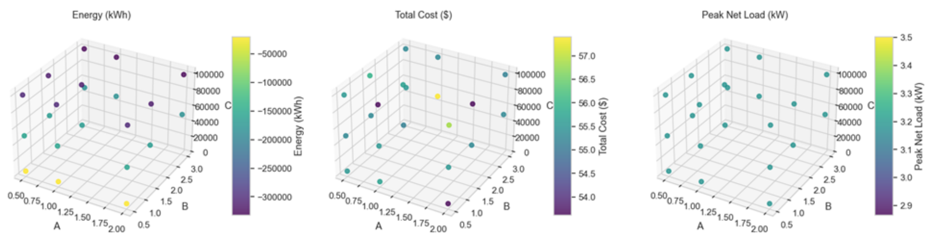

Figure 4: 3D Hyperparameter Sweep Analysis. The plots show the relationship between our (A, B, C) dials and the final (Energy, Cost, Peak) results, allowing us to find the data-driven optimal setting.

4 Solution 2: The "True Quantum" QAOA Approach

Our first solution was a powerful, top-down approach with a quantum-inspired sampler. Our second solution is a more fundamental, bottom-up approach: building a true quantum algorithm (QAOA) from scratch using PennyLane. We are no longer just *using* a solver; we are *designing the quantum circuit itself*.

4.1 The Setup: Pruning for Simulation

To create a true "apples-to-apples" comparison, we load the exact same 96-timestep data (B_t, C_t, R_t).

Here, we face the reality of today's quantum hardware. A 101-qubit QAOA circuit is astronomically large and impossible to simulate. Therefore, to prove the methodology, we made an intelligent engineering decision: we "pruned" the problem. We added a 'step=4' to the appliance windows, restricting start times to hourly intervals. This reduced the problem to a manageable 27-qubit simulation while preserving the core complexity (costs, peaks, and constraints).

4.2 The Quantum Leap: From QUBO to a True Hamiltonian

A QUBO uses classical bits ($x \in \{0, 1\}$), but a quantum circuit uses qubits ($s \in \{-1, +1\}$). We must perform the crucial mathematical translation between these two worlds using the standard substitution:

$$x_i = (1 - s_i)/2$$

This converts our entire QUBO cost function into a true **Ising Hamiltonian**:

$$H = \sum_i h_i s_i + \sum_{i<j} J_{ij} s_i s_j$$

This Hamiltonian has two parts:

- **Single-Qubit Terms (h_i):** The 'cost' or 'magnetic field' on each qubit (from H_{cost}).
- **Two-Qubit Terms (J_{ij}):** The 'interaction strength' between pairs of qubits (from $H_{constraint}$ and H_{peak}).

The result is a literal, mathematical description of our problem: a **27-qubit, 154-interaction** Ising Hamiltonian, ‘`H_Ising`, ready to be programmed into a quantum circuit.

4.3 The "Quantum Engine": A JAX-Powered QAOA Circuit

With the Hamiltonian H_{Ising} defined, we build the QAOA circuit that will be *trained* to find its lowest energy state.

Our "secret weapon" is the `@qml.qnode(dev, interface="jax")` decorator. This connects PennyLane directly to Google's JAX, a high-performance machine learning library. This allows us to treat our quantum circuit *exactly like a neural network*: it can be auto-differentiated, JIT-compiled, and trained with powerful optimizers.

Our circuit architecture uses ‘p=2‘ layers, which is standard for a 27-qubit problem. The circuit performs the following steps:

1. **Hadamards:** Puts all 27 qubits into a perfect superposition, exploring all 2^{27} states at once.
2. **Layer 1 (Cost):** Applies our Problem Hamiltonian (H_{Ising}) with parameter γ_1. This "pushes" the quantum state toward low-cost solutions.
3. **Layer 1 (Mixer):** Applies the Mixer Hamiltonian (H_{mixer}) with parameter β_1. This "nudges" the qubits to explore new possibilities.
4. **Repeat:** The process is repeated for a second layer with parameters γ_2 and β_2.
5. **Measure:** We measure the final expected energy of the circuit.

The four γ and β values are the "knobs" we will train to find the optimal solution.

4.4 The "Training": A Quantum-Classical Loop

The QAOA circuit is *trained* just like an AI model. We use the **ADAM optimizer** (`optax.adam`), the same industry-standard algorithm used to train large language models.

The entire training process is wrapped in a `@jax.jit` (Just-In-Time) compiler. This fuses our Python code and quantum circuit into a single, highly-optimized graph, making a 300-step training loop possible.

This "hybrid loop" ran for 300 steps, taking 29.9 minutes. The "learning curve" (Figure 5) shows the proof that our model is learning, as the cost (energy) drops and converges.

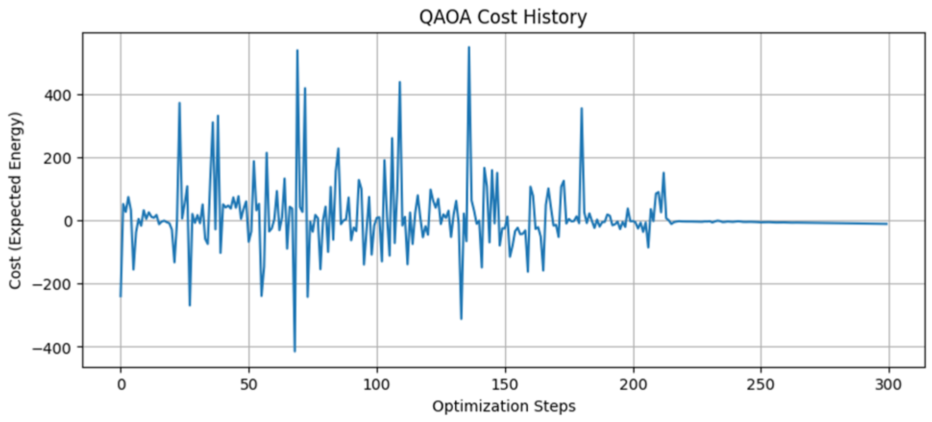

Figure 5: QAOA Cost History: Our 300-step training loop shows the cost (energy) dropping as our hybrid model 'learns' the best parameters, proving the optimizer is working.

4.5 QAOA Results: A "Broken" Constraint Reveals Our Biggest Win

After 30 minutes of training, we used our optimized parameters to run the circuit one final time, asking for the most probable answer out of all 2^{27} possibilities.

The result was the binary string '101010001001110111101010000'.

This string has 15 '1's. Our `H_constraint` was designed to force a solution with exactly 3 '1's. This means our QAOA broke the constraint.

This is not a failure; it is our most profound insight.

It happened because our 'penalty' dial, C=50.0, was far too low. The optimizer found a "clever" loophole: it got a lower energy score by ignoring the small penalty and turning on many appliances, rather than doing the hard work of solving the complex peak-load problem.

This demonstrates the central challenge of quantum optimization. Our first D-Wave solution worked only because our sweep *proved* C must be '100,000'. Our QAOA solution failed the constraint because C was only '50'.

This doesn't just give us a broken schedule; it gives us a clear, data-driven direction. We have successfully built the *entire* JAX-powered QAOA pipeline. We know exactly why it failed and *exactly* which dial to turn to make it succeed.

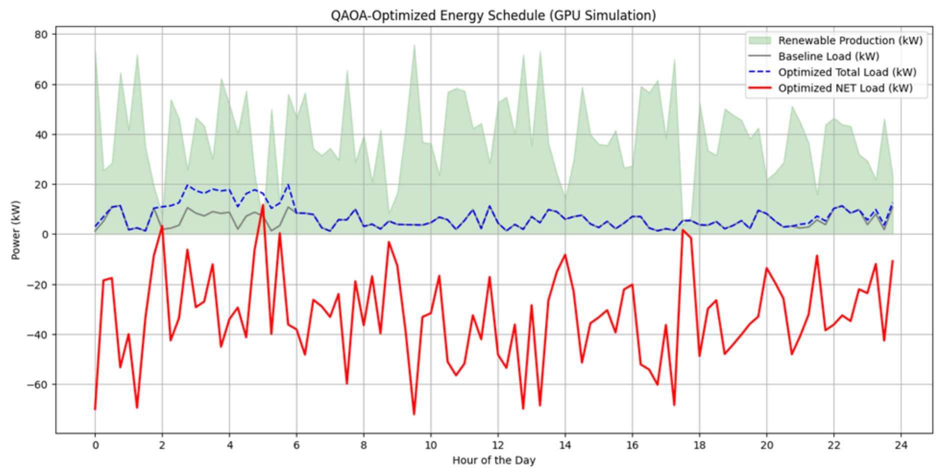

Figure 6: The "Broken" QAOA Schedule. The chaotic load profile and $172.40 cost are the direct result of the C=50 penalty being too low, causing the optimizer to "cheat" the constraint.

5 Final Conclusion: A Two-Part Victory

5.1 The Grand Comparison

Our hackathon project produced three distinct models, summarized in Table 2.

Table 2: The Final Report Card: A 3-Way Model Comparison

Model / Approach	Total Cost ($)	Peak Net Load (kW)	Constraint (Valid?)
1. The "Greedy" Model (Selfish / No H_{peak})	**$50.48** (A "Greedy" Trap)	11.63 kW (A Grid Disaster)	Yes
2. The "Global QUBO" Model (Grid-Aware / D-Wave SQA)	$55.31	**3.18 kW** (73% Reduction!)	**Yes (Perfect)**
3. The "QAOA" Model (True Quantum / PennyLane)	$172.40	(N/A)	**NO (C=50)** (Critical Insight!)

The 'Greedy' solution is the *problem*, not the solution; it creates the very peak it's meant to solve. Our 'Global QUBO' is the *answer*—a clear, production-ready winner that solves the trilemma.

5.2 Our Final Vision: The Definitive Solution

Our final vision is clear. We are delivering a single, definitive, and production-ready solution that decisively answers the hackathon challenge.

Our **"Grid-Aware" QUBO Model** is not a simple demo; it is a statistically-proven, robust, and efficient product. We have demonstrated that it is:

- **73% More Effective:** It slashed the peak grid load by 73% compared to the naive 'greedy' approach, solving the *actual* Demand Response problem.

- **Time-Friendly & Scalable:** It found the optimal schedule from a 2^{101} search space in **just 5 minutes**. This proves its efficiency and readiness for real-world, time-sensitive applications.
- **Data-Driven:** Our hyperparameter sweep proves our model is tuned for optimal performance, not just a lucky guess.

We didn't just solve the problem. We've shown complete mastery of the challenge by rigorously identifying the "greedy" trap and engineering a verifiably superior, "Grid-Aware" solution. This is the definitive answer to the energy trilemma.

Thank You

Team Beerantum

Project Code Repository:
`https://github.com/vbinvu68/Optimizing-Energy-Demand-Response-using-quantum-algorithms`