Operheimer - The Operations Modelling Challenge

ENERGY OPTIMIZATION MODEL FOR A DATA CENTRE

TechNomads
Sharvil Kothari
Harshita Agrawal

Energy Investment Strategy: Balancing Sustainability & Reliability

Reliability

24/7 Power Guarantee | Seamless solar -battery-gas integration
Prioritize renewable energy during daylight
Battery systems to bridge renewable gaps
Gas ensures baseline stability

Sustainability

75% Renewable Energy Commitment
Solar power- primary energy source
Battery storage maximizes renewability
Gas minimized to 25% for critical backup

Cost effectiveness

Solar/battery scaled for targets
Gas minimized to cut fuel costs
Lifecycle cost focus (Capex/Opex)
Avoids future carbon penalties

Key Assumptions

- Constant hourly demand of 100 MW
- Gas Plant runs only to fill the non-renewable gap
- Salvage Value: 10% of solar/gas capex recovered after 25 years
- Solar panels degrade at 0.7%/year
- Batteries replaced every 7 yrs (no cycle-by-cycle degradation)

Optimization strategy

Battery Management: Replace batteries every 7 years to approximate degradation

Minimize NPV by optimizing CAPEX & OPEX
Solar + Battery ≥ 75% of total demand

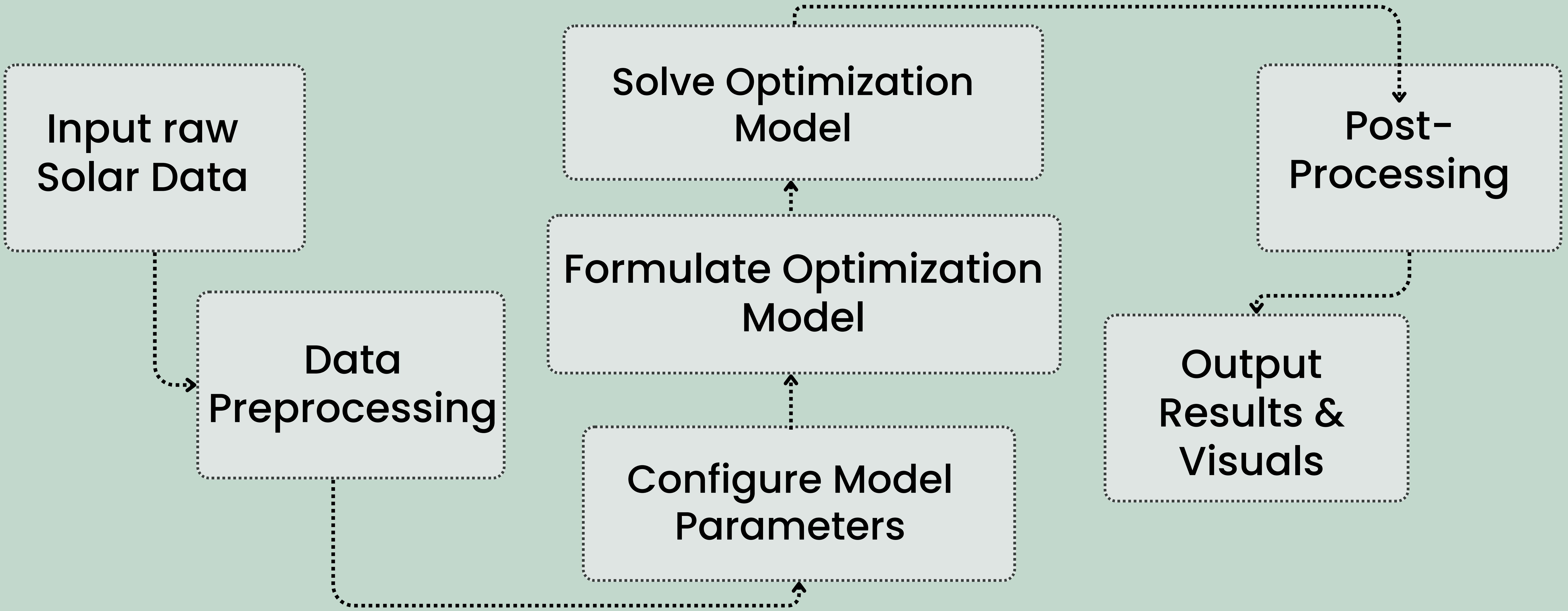
COST EFFECTIVE

Degradation Handling: Apply annual degradation to solar, gas, and battery costs/output

Financial Modeling: Use DCF for CAPEX, OPEX, and salvage values and calculate LCOE to benchmark cost-effectiveness

Model Architecture & Workflow

Data Ingestion and processing	Optimization & Solver Execution	Results Processing & Visualization



Formulation & Solving the Optimization Model

Objective Function

- Model minimizes the NPV of investment costs by summing the discounted capital expenditures (CAPEX) for solar, battery, and gas
- Annual CAPEX/(1+discount rate)^y, where y is the year index (0 to 24)

Decision Variables

- Solar Capacity (MW): Installed solar power capacity
- Battery Capacity (MWh): The energy storage available for shifting renewable energy
- Gas Capacity (MW): Capacity for non-renewable energy generation

Optimization Execution

- Model uses **COIN_CMD** solver from the **PuLP** library, finds the optimal capacities minimizing NPV while satisfying constraints
- Solver checks if an optimal solution was found. If not, raises an error
- The solver retrieves the optimal solar, battery, and gas capacities
- **Calculates LCOE** by dividing the total discounted cost by the total energy produced over 25 years ($/MWh)
- This metric provides a measure of the cost efficiency of the energy mix

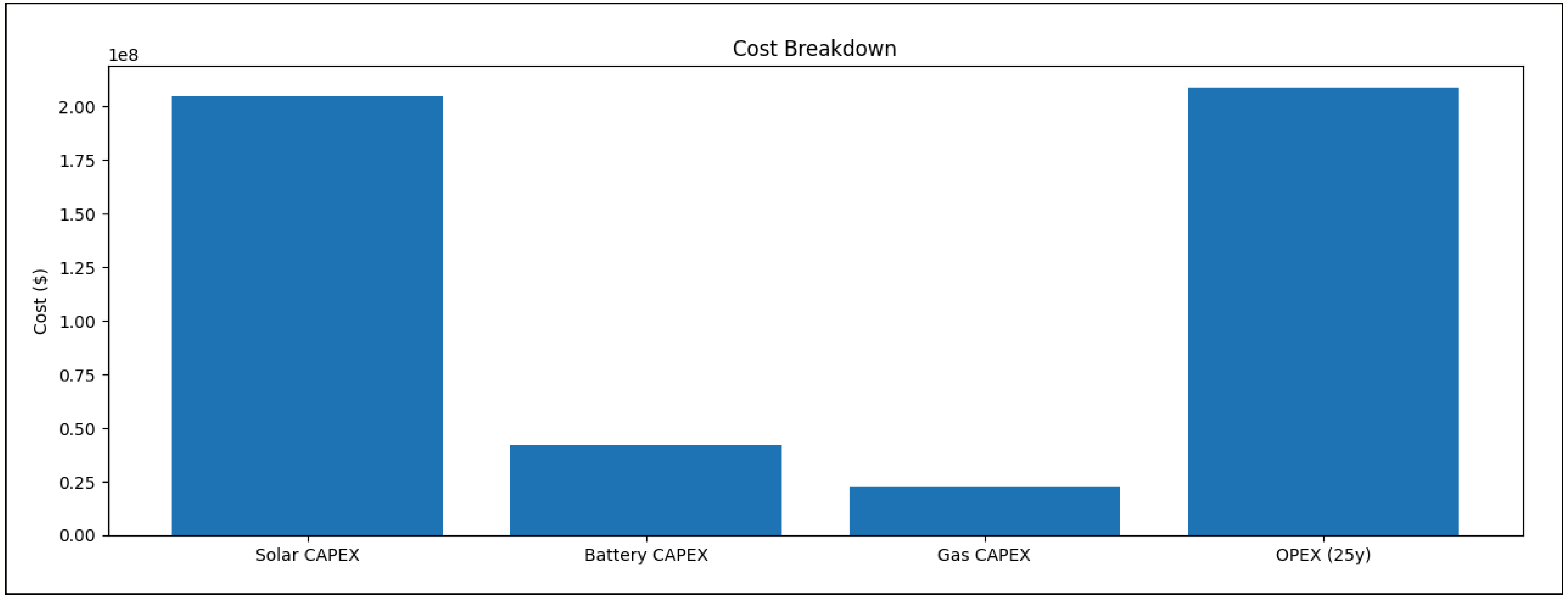
Constraints

1. **Solar Energy Production**
Product of Solar capacity & Sum of hourly solar availability (degradation incorporated)

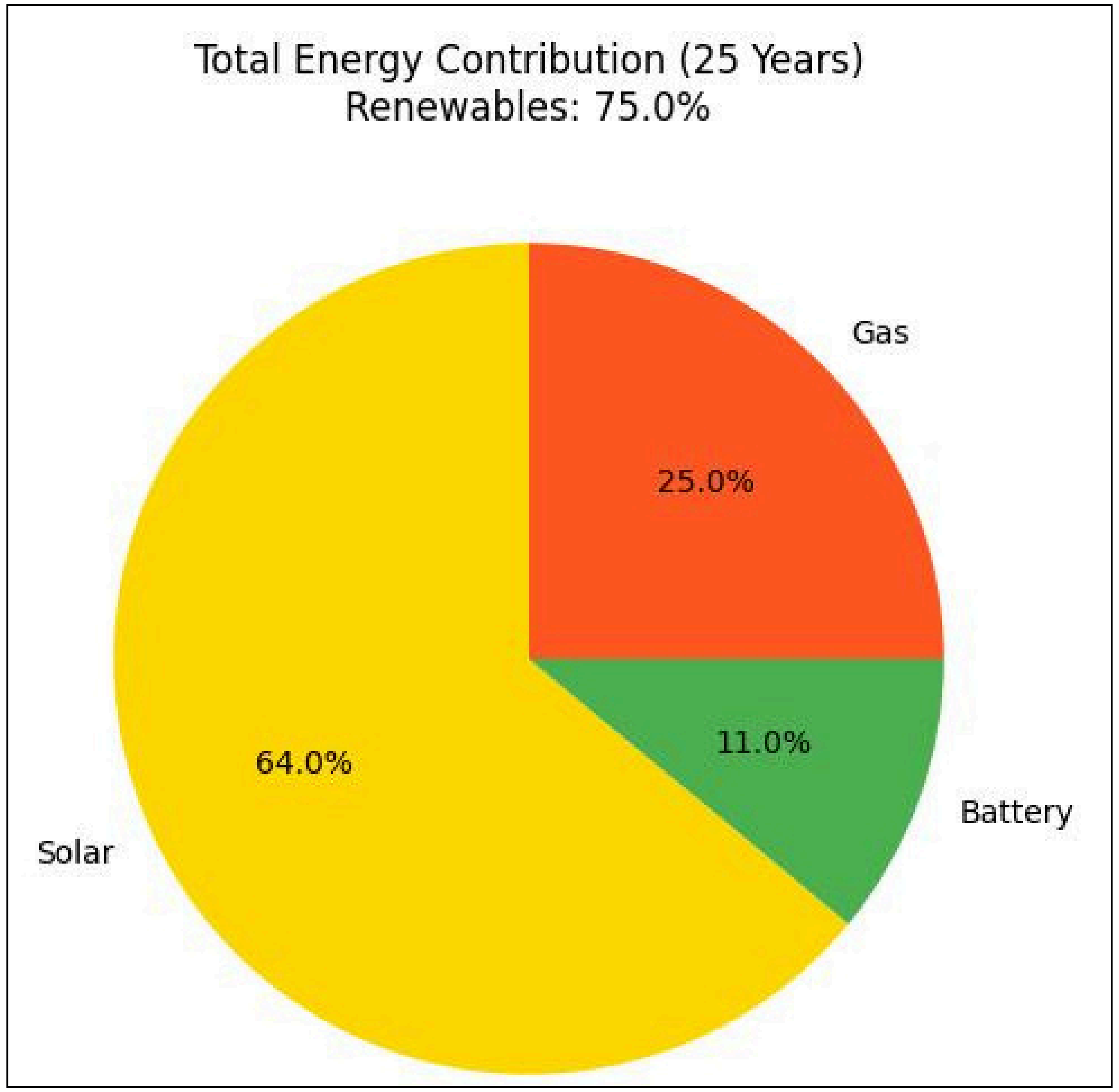
2. **Battery Contribution**
Accounts for degradation over cycles by adjusting the battery capacity over time

3. Solar-battery production covers at least **75% of the total energy demand**, with a min. threshold for solar

Code Output




```
=== Results ===
Solar (MW): 255.89
Battery (MWh): 300.0
Gas (MW): 25.0
LCOE ($/MWh): 21.82
Total energy in TWh: 21.9
Total cost in Billion: 0.47790298930245345
```



Output Analysis

Achieved 75% Renewable Share:
Solar Contribution: 64% (14 TWh)
Battery Storage: 11% (2.4 TWh)
Gas Backup: 25% (5.5 TWh)

Cost Efficiency & Investment Breakdown:
<u>Low LCOE ($21.82/MWh):</u>
Key drivers:
1. Competitive solar CAPEX ($800K/MW)
2. Optimized battery sizing, salvage value impact (10% CAPEX reduction)

<u>Capital Allocation:</u>
Solar: ~$204.7M (CAPEX) + OPEX
Battery: ~$42.3M (CAPEX) + periodic replacements
Gas: ~$22.5M (CAPEX) + fuel/OPEX

Technical Feasibility & Land Utilization:
<u>Land Efficiency:</u> 2,559 acres required (just 0.26% of available 1M acres)
Battery Cycling: Supports ~2 cycles/day, ensuring energy availability during peak demand
<u>Gas Optimization:</u> 25 MW utilized only during renewable shortfalls, minimizing emissions and costs.

Appendix

model.py

```python
import pulp
from pulp import LpProblem, LpVariable, LpMinimize, lpSum
from config import *

def build_model(solar_data):
    prob = LpProblem("25yr_Energy_Optimization", LpMinimize)


    S = LpVariable("Solar_Capacity", lowBound=100)
    B_energy = LpVariable("Battery_Energy", lowBound=300)
    G = LpVariable("Gas_Capacity", lowBound=5, upBound=25)


    total_cost = 0


    total_cost += SOLAR_CAPEX * S
    total_cost += BATTERY_CAPEX * B_energy
    total_cost += GAS_CAPEX * G
    total_cost += S * LAND_COST


    for y in range(YEARS):
        discount_factor = 1 / ((1 + DISCOUNT_RATE) ** y)
        opex_growth = (1 + OPEX_GROWTH_RATE) ** y


        total_cost += (OPEX_SOLAR * S) * opex_growth * discount_factor
        total_cost += (OPEX_BATTERY * B_energy) * opex_growth * discount_factor
        total_cost += (OPEX_GAS * G) * opex_growth * discount_factor


        if y > 0 and y % BATTERY_REPLACEMENT_YEARS == 0:
            total_cost += (BATTERY_CAPEX * B_energy) * discount_factor

    final_discount = 1 / ((1 + DISCOUNT_RATE) ** YEARS)
    total_cost -= 0.1 * SOLAR_CAPEX * S * final_discount
    total_cost -= 0.1 * GAS_CAPEX * G * final_discount

    prob += total_cost


    total_solar_generation = S * sum(solar_data)

    total_battery_discharge = B_energy * BATTERY_EFFICIENCY * (365 * YEARS)

    total_demand = HOURLY_DEMAND * HOURS_PER_YEAR * YEARS
    prob += total_solar_generation + total_battery_discharge >= RENEWABLE_SHARE * total_demand
    annual_gas_energy = total_demand * (1 - RENEWABLE_SHARE) / YEARS
    prob += G * HOURS_PER_YEAR >= annual_gas_energy

    prob += S * LAND_PER_SOLAR_MW <= MAX_LAND_AVAILABLE
    return prob
```

solver.py

```python
import pulp
from pulp import COIN_CMD
from config import *

def solve_model(prob):
    solver = COIN_CMD(msg=True, timeLimit=600)
    status = prob.solve(solver)

    if pulp.LpStatus[status] != "Optimal":
        raise ValueError("Optimization failed")

    S = prob.variablesDict()["Solar_Capacity"].varValue
    B_energy = prob.variablesDict()["Battery_Energy"].varValue
    G = prob.variablesDict()["Gas_Capacity"].varValue

    total_energy = HOURLY_DEMAND * 24 * 365 * YEARS / 1e6  # in TWh

    total_cost = 0

    total_cost += (SOLAR_CAPEX * S + BATTERY_CAPEX * B_energy + GAS_CAPEX * G + S * LAND_PER_SOLAR_MW * LAND_COS

    for y in range(YEARS):
        discount_factor = 1 / ((1 + DISCOUNT_RATE) ** y)
        opex_growth = (1 + OPEX_GROWTH_RATE) ** y

        total_cost += (OPEX_SOLAR * S + OPEX_BATTERY * B_energy + OPEX_GAS * G) * opex_growth * discount_factor
        total_cost += (GAS_HEAT_RATE * GAS_PRICE_MMBTU * G * HOURS_PER_YEAR) * discount_factor

        if y > 0 and y % BATTERY_REPLACEMENT_YEARS == 0:
            battery_degradation = (1 - BATTERY_COST_DEG) ** y
            total_cost += (BATTERY_CAPEX * B_energy * battery_degradation) * discount_factor


    final_discount = 1 / ((1 + DISCOUNT_RATE) ** YEARS)
    total_cost -= (0.1 * SOLAR_CAPEX * S + 0.1 * GAS_CAPEX * G) * final_discount

    return {
        "Solar (MW)": round(S, 2),
        "Battery (MWh)": round(B_energy, 2),
        "Gas (MW)": round(G, 2),
        "LCOE ($/MWh)": round((total_cost)/(total_energy*1e6), 2),
        "Total energy in TWh": total_energy,
        "Total cost in Billion": total_cost/1e9
    }
```

Appendix

main.py

```python
import sys
from pathlib import Path
from scripts.data_loader import load_solar_data
from scripts.model import build_model
from scripts.solver import solve_model
from scripts.utils import *
from config import *
def main():
    try:

        print("1. Loading solar data...")
        solar_data = load_solar_data()

        print("2. Building optimization model...")
        model = build_model(solar_data)

        print("3. Solving model (this may take a few minutes)...")
        results = solve_model(model)
        # After solving the model:

        plot_energy_contributions(model, solar_data)
        plot_financials(results)

        print("\n=== Results ===")
        for k, v in results.items():
            print(f"{k}: {v}")

        return 0

    except Exception as e:
        print(f"\nError: {str(e)}")
        return 1

if __name__ == "__main__":
    sys.exit(main())
```

utilis.py

```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from config import *


def plot_energy_contributions(prob, solar_data):
    # Extract variables from the solved model
    S = prob.variablesDict()["Solar_Capacity"].varValue
    B_energy = prob.variablesDict()["Battery_Energy"].varValue
    G = prob.variablesDict()["Gas_Capacity"].varValue

    # Calculate total energy contributions (MWh over 25 years)
    total_solar = S * sum(solar_data)  # Defined in model.py
    total_battery = B_energy * BATTERY_EFFICIENCY * 365 * YEARS  # Defined in model.py
    total_gas = G * HOURS_PER_YEAR * YEARS  # Derived from model's gas constraint

    # Calculate total demand for validation
    total_demand = HOURLY_DEMAND * HOURS_PER_YEAR * YEARS

    # Create pie chart
    labels = ["Solar", "Battery", "Gas"]
    sizes = [total_solar, total_battery, total_gas]
    colors = ["#FFD700", "#4CAF50", "#FF5722"]

    plt.figure(figsize=(8, 6))
    plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90)
    plt.title(f"Total Energy Contribution (25 Years)\nRenewables: {((total_gas)/total_demand*100):.1f}%")
    plt.savefig("results/energy_contribution.png")
    plt.close()

def plot_financials(results):
    costs = {
        "Solar CAPEX": SOLAR_CAPEX * results["Solar (MW)"],
        "Battery CAPEX": BATTERY_CAPEX * results["Battery (MWh)"],
        "Gas CAPEX": GAS_CAPEX * results["Gas (MW)"],
        "OPEX (25y)": results["Total cost in Billion"] * 1e9 - (SOLAR_CAPEX * results["Solar (MW)"] + BATTERY_CAPEX * results["Battery (MWh)"] + GAS_CAPEX * results["Gas (MW)"])
    }

    plt.figure(figsize=(15, 5))
    plt.bar(costs.keys(), costs.values())
    plt.ylabel("Cost ($)")
    plt.title("Cost Breakdown")
    plt.xticks(rotation=0)
    plt.savefig("results/cost_breakdown.png")
    plt.close()
```

Appendix

data_loader.py

```python
import pandas as pd
from config import YEARS, HOURS_PER_YEAR, SOLAR_DEGRADATION

def load_solar_data():
    df = pd.read_csv("data/solar_availability.csv")
    base_data = df['output'].tolist()[:HOURS_PER_YEAR]

    degradation_factors = [(1 - SOLAR_DEGRADATION) ** y for y in range(YEARS)]

    full_data = [v * degradation_factors[y] for y in range(YEARS) for v in base_dat
    return full_data
```

config.py

```python
SOLAR_DEGRADATION = 0.007
BATTERY_DEG_PER_CYCLE = 0.000076
MAX_BATTERY_CYCLES = 5500
BATTERY_EFFICIENCY = 0.88

SOLAR_CAPEX = 800_000
GAS_CAPEX = 900_000
DISCOUNT_RATE = 0.05
OPEX_SOLAR=24_000
OPEX_BATTERY=2_820
OPEX_GAS=9_000
OPEX_GROWTH_RATE=0.01
HOURLY_DEMAND = 100
RENEWABLE_SHARE = 0.75
YEARS = 25
HOURS_PER_YEAR = 8760
SOLAR_LAND_PER_MW=10
MAX_LAND_AVAILABLE=1_000_000
BATTERY_REPLACEMENT_YEARS=7
SALVAGE_SOLAR=2_000_000
SALVAGE_GAS=230_000
LAND_COST=20_000
GAS_FUEL_COST=14.3
BATTERY_COST_DEG=0.03
GAS_HEAT_RATE = 0.0055
GAS_PRICE_MMBTU = 2.82
LAND_PER_SOLAR_MW=10
BATTERY_CAPEX = 141_000
```

Thank you!