CO-SIMULATION Project

Professor: Prof. Vincent Chevrier, Prof. Netter Denis

Co-Simulation of a Hybrid Renewable Energy System for Reliable District Heating in Cold Climates

A Case Study in Sweden

MAY 2025

Group:
Rashida Olomowewe
Walmy Fernández
Oluwajoba Elisalome Oyefusi
Sofia farakhutdinova
Morris Nyantee
Dang Chuong Ta

Master ENERGIE - Decentralized Smart Energy Systems 2

Table of Contents

1. Introduction

In cold-climate countries like Sweden, heating can account for up to 70% of total energy consumption in residential buildings. Ensuring a reliable heat supply; especially during winter, is critical, but traditional systems are often reliant on fossil fuels. This creates both environmental and operational challenges, particularly under variable weather conditions.

This project investigates how to maintain a fixed water outlet temperature in a district heating system using a hybrid setup that includes photovoltaic (PV), wind turbine, battery storage, and a gas turbine. The goal is to supply the required thermal power continuously, even as weather conditions change, while minimizing reliance on the gas turbine.

The system is implemented using a co-simulation approach with a real-time feedback loop. Each energy subsystem is modeled in OpenModelica, exported as a Functional Mock-up Unit (FMU), and integrated in Python. A centralized Python-based controller reads system outputs, such as battery SOC, renewable generation, and heater demand, and sends control signals to dynamically allocate energy resources.

By closing the loop between sensing and actuation, the system can adapt in real time to maintain thermal demand, prioritize low-carbon sources, and only activate the gas turbine when needed. This feedback-driven architecture allows for scalable, modular, and realistic simulation of smart energy systems under winter scenarios.

1.2 System Architecture

This project develops a co-simulated district residential heating microgrid designed specifically for the harsh winter conditions in Sweden. It consists of five interconnected subsystems, each exported as a FMU from OpenModelica and simulated together in Python:

1. Photovoltaic (PV) Subsystem

- Generates electricity based on irradiance data.
- It is exported as an FMU and used in co-simulation to provide daily variable input power.

2**. Wind Turbine Subsystem**

- Complements solar generation by converting wind speed into electrical power.
- It captures the nonlinear relationship between wind speed and mechanical power.
- This subsystem is simulated as an FMU with inputs from either real-time or synthetic wind profiles.

3.**Gas Turbine Subsystem**

- Acts as a controllable source of heat and power.
- It supplies both electricity and heat, particularly during peak demand periods or when renewable energy sources are low.

4. **Battery Storage Subsystem**

- Stores excess renewable energy and provides power during shortages.

- It monitors the State of Charge (SoC), efficiency losses, and degradation behaviour.
- The FMU allows for flexible and responsive integration with Python-based logic.

5. **Supervisory Control System**
   - Coordinates all FMUs through a master algorithm implemented in Python.
   - It utilizes real-time information (e.g., SoC, load demand, and weather inputs) to:
     - Prioritize renewable energy sources.
     - Activate the gas turbine as needed.
     - Manage battery charging and discharging.
   - The system implements a hybrid control strategy that combines rule-based methods with scheduling logic.
   - It communicates with each FMU through variable updates and event triggers.

It is implemented to react dynamically to changes in demand, weather, and generation forecasts, making the system both efficient and resilient. The following sections explain in depth the design of each of the subsystems.

1.3 Use Case

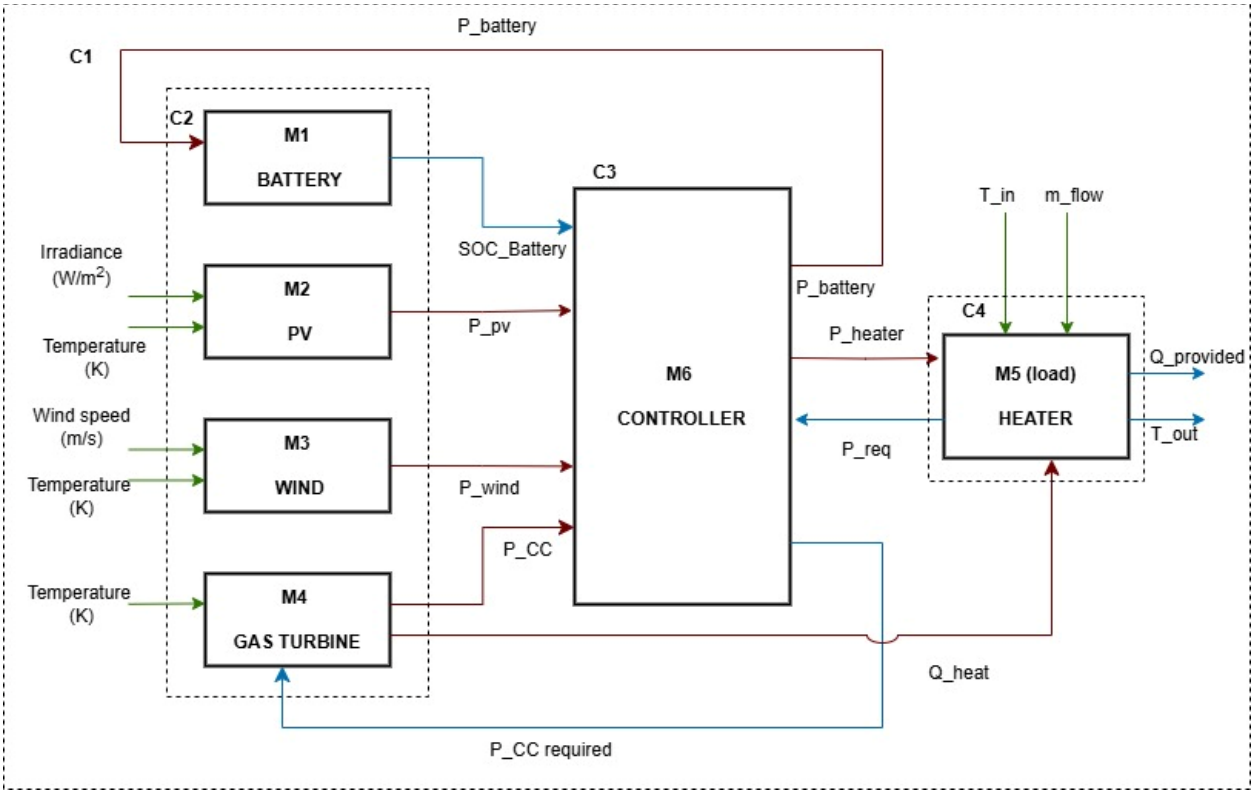

Figure 1. Block diagram showing Hybrid electrical system with its inputs & outputs

1.4 Location and weather data description

To evaluate system performance under realistic operating conditions, the simulated heating system is placed in a Swedish winter context, a setting where centralized district heating is widely used to

meet high residential heat demand. The goal is to validate the system's ability to maintain a fixed outlet temperature while dynamically responding to changes in weather-dependent renewable energy availability.

To test the hybrid microgrid's ability to maintain a constant heater outlet temperature under varying weather conditions, a three-day simulation scenario was constructed. Each day represents a different winter weather regime: sunny, rainy, and snowy, to challenge the system's adaptability and energy management strategy.

1.5 Weather Data Source and Preparation

The environmental data was sourced from the Global Solar Atlas and similar public meteorological datasets. The Figure 2. below presents the simulated weather conditions used as input for the three-day winter scenario. These environmental profiles, sampled at 15-minute intervals, include solar irradiance, wind speed at hub height (110 m), and ambient temperature.

- Day 1 (Sunny Conditions): Solar irradiance peaks above 300 W/m² in a smooth diurnal cycle. Wind speed follows a sinusoidal profile around 9 m/s, while temperatures remain mild, averaging around 2 °C.
- Day 2 (Rainy Conditions): Solar input drops to ~200 W/m², with frequent wind gusts pushing speeds up to 19 m/s. Temperatures fall to around 0 °C, reflecting unstable atmospheric conditions.
- Day 3 (Snowy Conditions): Heavy cloud cover reduces irradiance to below 100 W/m². Wind stabilizes at ~5 m/s, and temperatures drop sharply to an average of –8 °C.

Additional internal parameters, such as yaw angle, blade pitch, and generator torque, were modeled separately to simulate turbine control behavior and aerodynamic response under each scenario.

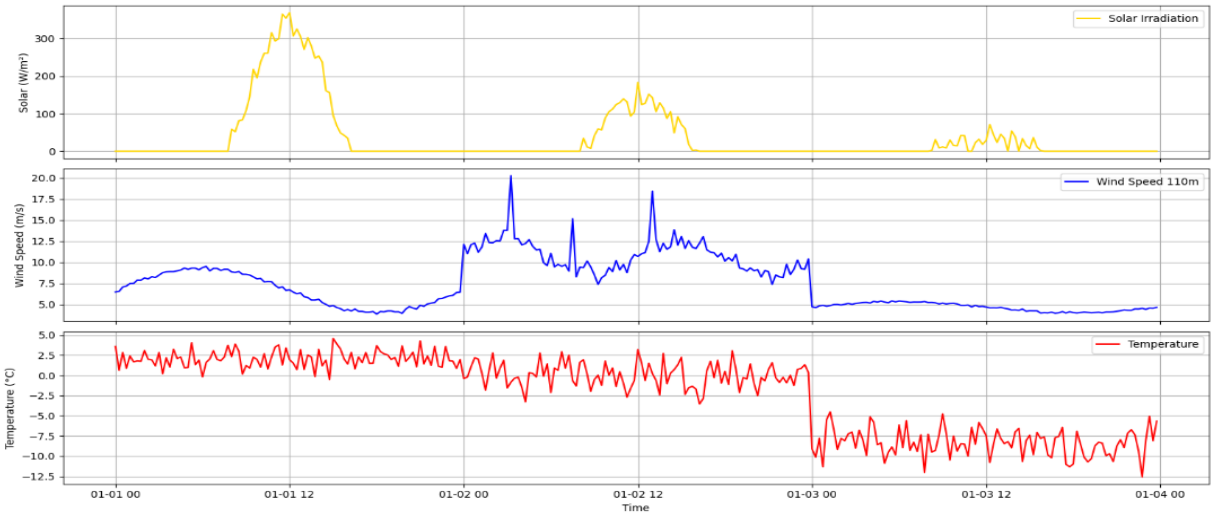

Figure 2. Overview of simulated meteorological inputs for a three-day period, including solar irradiance, wind speed at 110 m, and ambient temperatur.

2.Methodology

Each component of the hybrid energy system; PV, wind turbine, gas turbine, battery, heater, and controller was modeled independently in OpenModelica and exported as an FMU using the FMI standard. These FMUs were then tested individually and then integrated into a Python-based co-simulation framework. This section presents the objective, key parameters, modeling approach, and validation results for each subsystem prior to full system integration.

2.1 Gas Turbine

The decentralized energy system is composed of renewable generation units as PV, wind turbines, and battery storage. While these components provide sustainable and low-emission power, they are subject to significant fluctuations due to their dependence on weather conditions and day-night variation. To address the variability and ensure continuous energy availability, a gas turbine was added as a backup source.

The system operates under the supervision of a central controller, which dynamically assesses the overall energy balance and determines the required electrical output from the gas turbine. This control mechanism ensures optimal coordination with other subsystems such as battery storage, PV and wind turbines.

The gas turbine is integrated within a combined cycle configuration, enabling simultaneous production of electricity and useful thermal energy. Waste heat from the turbine exhaust is recovered and utilized, improving total system efficiency. This strategy reduces fuel consumption and enhances the energy utilization factor.

For the project the following assumptions were made:

1. Steady-state assumption without considering compressor/turbine dynamics.
2. Operational constraints are used to prevent unrealistic operating regimes
3. Implicitly models Heat Recovery Steam Generator (HRSG) through constant efficiency ratio

Table 1. Key parameters for modelling.

Parameter	*Value*
η_{el}	0.35
η_{therm}	0.45
P_{min}	350 kW
P_{max}	1000 kW

Table 2. Input and Outputs.

Parameter	*Type*	*Direction*	*Unit*	*Description*
P_{CCreq}	Real	Input	W	Requested electrical power
T_{amb}	Real	Input	K	Ambient air temperature
P_{CC}	Real	Output	W	Actual electrical power output
Q_{heat}	Real	Output	W	Recovered thermal energy output

The turbine receives its power command (P_CCreq) from a controller that balances generation with load demand across all energy sources. The following logic was implemented:

To ensure the power requested is within the turbine's operating range:

$$P_{requested} = \min\left(P_{max}\max\left(P_{min,}P_{CCreq}\right)\right.$$

If $P_{CCreq} > 0$, the Power output is calculated in the following way:

$$P_{CC} = P_{requested}\left(1 - 0.0015\cdot\left(T_{amb} - 298\right)\right)$$

The recovered heat based is calculated based on efficiencies:

$$Q_{heat} = \left(\frac{\eta_{thermal}}{\eta_{electrical}}\right)P_{CC}$$

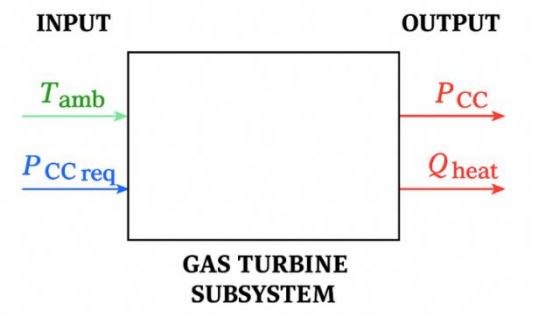

Figure 3. Block diagram of the gas turbine.

2.1.1 Testing model in OpenModelica

To evaluate the performance of the gas turbine model, a test was implemented in OpenModelica. For testing, three representative cases were simulated:

Case 1: A requested power of 400 kW (within limits) resulted in a slightly derated output (~394 kW) due to elevated ambient temperature (308 K).

Case 2: A high request of 1.2 MW exceeded the maximum limit, and the output was correctly clamped to 1 MW.

Case 3: A low request of 200 kW triggered the minimum power constraint, with output held at 350 kW.

These cases validated the turbine's derating behavior and enforcement of operating boundaries.

The result for the Case 1 is shown below:

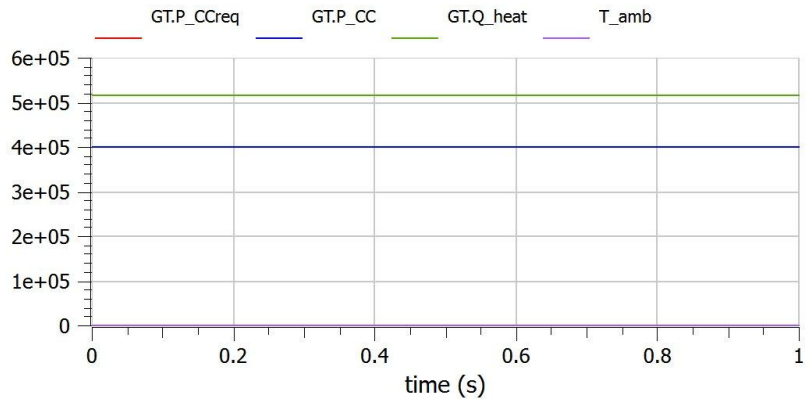

Figure 4. Simulation results for Case 1.

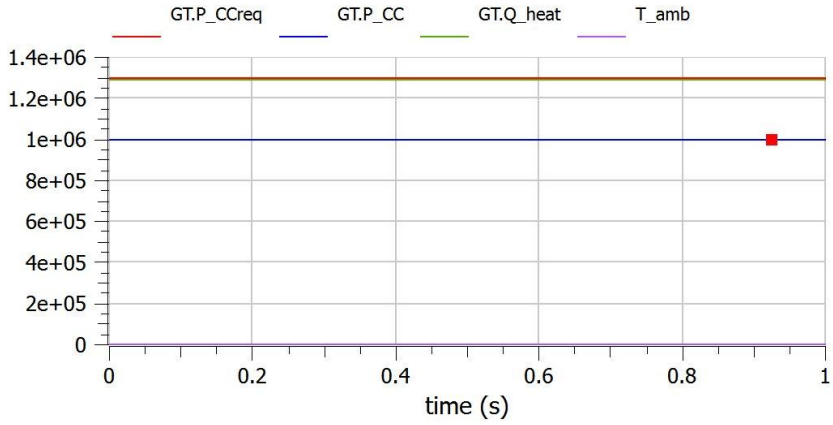

Figure 5. Simulation results for Case 2.

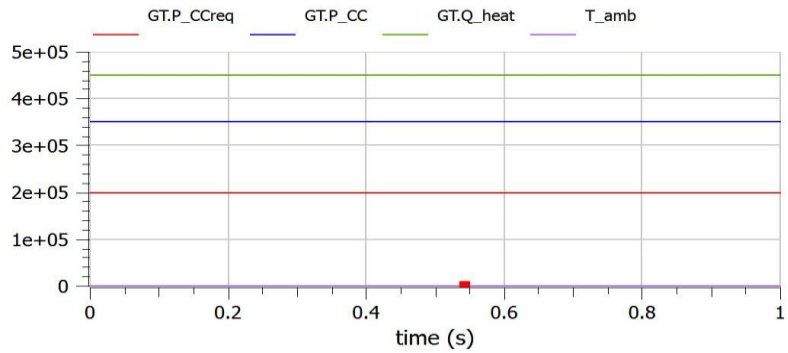

Figure 6. Simulation results for Case 3.

To make sure the Gas Turbine FMU works properly, a simple test function was created. It starts by initializing the model, then sets some realistic input values like the power request and ambient temperature. After running the simulation for a short time, the key outputs electrical and thermal power are printed out. This test helps confirm that the turbine behaves correctly before using it in the full system setup.

```
def test_gasturbine():
    print("\n=== Gas Turbine self-test ===")

    G = myFMU(gasturbinefmu)
    G.init(0.0, [])

    G.set(['P_CCreq', 'T_amb'], [400000.0, 308.0])

    G.doStep(0.0, 60.0)

    print("P_CC       [W] :", G.get('P_CC'))
    print("Q_heat     [W] :", G.get('Q_heat'))

    G.terminate()
```

Figure 7. Python test code for the gas turbine model.

2.2 PV System

This section focuses on the modeling and simulation of a photovoltaic (PV) subsystem in a hybrid renewable energy microgrid that also includes wind and gas turbine units. The primary function of the PV unit is to supply electrical power to an electric heater and simultaneously provide real-time control signals (e.g Power output) to the central control system.

The PV subsystem**:**

- Directly powers the electric heater.
- Sends power availability/status (P_pv) to central control unit.

This output (P_pv) is a control signal used by the central system to manage energy distribution across sources.

The following table summarizes the key parameters, physical inputs, and power outputs of the PV model, grouped into power, physical, and control interfaces. In this case, control data is minimal. Therefore, the model is open-loop.

Table 3. PV output interface and control signal

Name	*Type*	*Direction*	*Value*	*Unit*	*Description*
Vpv	Real	Output	24	V	Output voltage from the PV model
Ipv	Real	Output	-	A	Otput current from the PV model
P_pv	Real	Output	-	W	Power output of the PV system used as control signal

Table 4. PV physical and input parameters:

Name	*Type*	*Direction*	*Values*	*Unit*	*Description*
Irr	Real	Input	-	W/m²	Real-time solar irradiance
Temp	Real	Input	-	°C	Real-time temperature
n_pv	parameter	Internal	10000	—	Number of PV panels
p_stc	parameter	Internal	300	W	Rated power under standard test conditions
irr_stc	parameter	Internal	1000	W/m²	Standard irradiance (reference)
t_stc	parameter	Internal	25	°C	Standard temperature
c_t	parameter	Internal	0.004	1/°C	Temperature coefficient of performance

Table 5. PV Internal efficiency parameter

Name	*Type*	*Direction*	*Value*	*Unit*	*Description*
PV_eff	Parameter	Internal	0.25	%	PV panel efficiency

2.2.1 Simulation, Control, and Graph

The simulation aimed to model the PV subsystem using simplified parameters while maintaining standardized behavior. The PV output power is calculated as:

$$P_{p_v} = n_{p_v} \cdot PV_{eff} \cdot P_{stc} \cdot (\frac{Irr}{irr_{stc}}) \cdot (1 - c_t \cdot (Temp - t_{stc}))$$

This expression calculates the real-time power output of the PV panel by scaling its rated STC (Standard Test Condition) power according to actual irradiance and temperature values. A fixed output voltage of 24 V is assumed, with current calculated as:

$$I_{pv} = (\frac{P_{pv}}{V_{pv}})$$

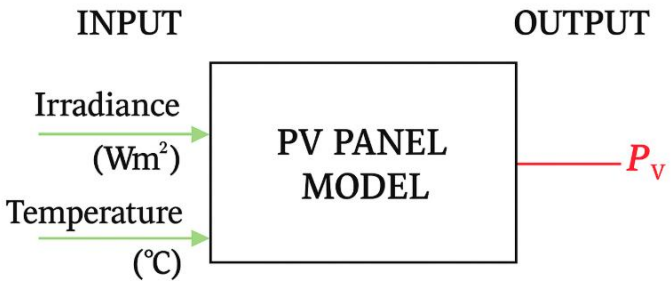

Figure 8. V subsystem interface within the hybrid co-simulation system.

2.2.2 OpenModelica Implementation

In this stage of the simulation workflow, the PV subsystem was modeled in OpenModelica based on the heater's specification. The model uses irradiance and temperature as inputs and computes electrical power (P_pv), current (I_pv), and voltage (V_pv) as outputs using simplified equations (see Equations 3.1 and 3.2). The figure below shows the simulation result under fixed environmental conditions (Irr = 800 W/m², Temp = 25 °C).

As expected, the output remains steady: power output stabilizes at 1 MW, and current at 100,000 A, assuming a constant voltage of 24 V. This confirms the model's correctness in computing energy output based on physical parameters. The output P_pv also represents the available solar power supplied to the central controller for co-simulation.

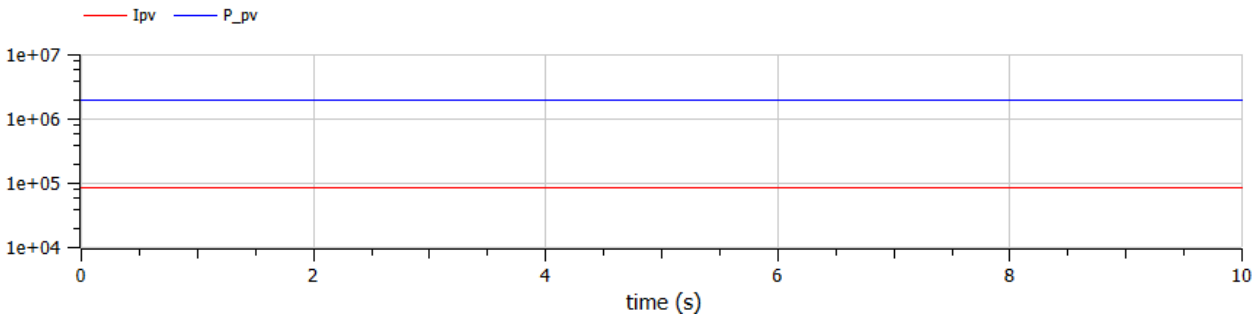

Figure 9. PV output for standalone model validation prior to integration.

2.2.3 Python FMU Testing

In Python, the PV FMU was tested using a get–set–doStep–get loop. It was initialized with fixed parameters such as panel efficiency and test conditions, then driven using dynamic irradiance and temperature data over a 72-hour period.

The simulation applied a time step of 0.1 s, and at each step, the FMU received real-time inputs, updated its internal state, and returned outputs: P_pv (power output), Vpv (voltage), and Ipv (current).

The results were plotted to confirm expected behavior power peaks aligned with high irradiance, and voltage/current followed typical diurnal patterns. This verified the FMU's response before integrating it into the full system, where it is managed by the central controller.

```python
def simulate():
    Fpv = myFMU(pvfmu)
    # Initial values
    dico = dict(n_pv=1, PV_eff=0.25,p_stc=300,irr_stc=1000, t_stc=25,c_t=0.004)
    startTime = 0.0
    stopTime = irr_df.index[-1]
    deltaT = 0.1
    initpv = ['PV_eff', 'n_pv', 'p_stc', 'irr_stc', 't_stc', 'c_t' ]
    inputpv = ['Irr', 'Temp']
    outputpv =  ['P_pv', 'Vpv', 'Ipv']
    # Initialize FMU with parameters
    Fpv.init(startTime, [(var, dico[var]) for var in initpv])
    # Time loop
    time = startTime
    tdata = []
    data = {v: [] for v in outputpv}
    while time <= stopTime:
        dico['Irr'] = float(irr_df.loc[round(time, 1), 'irr'])
        dico['Temp'] = float(Temp_df.loc[round(time, 1), 'temp'])
        Fpv.set(inputpv, dico)
        Fpv.doStep(time, deltaT)
        Fpv.get(outputpv, dico)
        tdata.append(time)
        for v in outputpv:
            data[v].append(dico[v])
        time += deltaT
    Fpv.terminate()
```

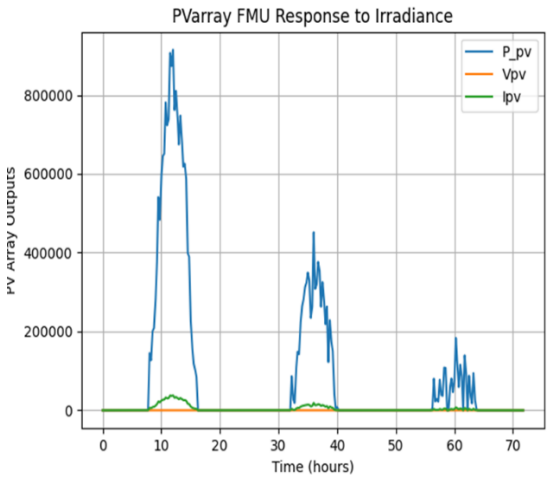

Figure 10. Python loop for simulating the PV FMU and Output plot under dynamic inputs

2.3 Wind Turbine Model

Wind power provides variable renewable electricity to support both the electrical heater and overall grid demand, reducing reliance on fossil-based generation.

Unlike PV, wind power contributes significantly during evening and night hours when solar is unavailable, ensuring continuous supply for both electrical heating and BESS charging. Its higher capacity factor makes it a more stable renewable backbone for supporting heat and electricity demands in the co-simulation.

2.3.1 Basic Implementation Using OpenModelica and Python

The basic wind power model, implemented in both OpenModelica and Python, uses a simplified steady-state approach. It takes key environmentally input wind speed, temperature, pressure, and humidity and applies standard turbine parameters like rotor radius, rated power, and cut-in/cut-out thresholds to calculate electrical power output. This model provides a static snapshot of turbine performance under given conditions, suitable for energy system simulations with limited dynamic behavior

Table 6. Input, Parameters and Output of Basic Wind Turbine

Input	Parameters (Wind turbine & Environment Constants)	Output
• Wind speed v_{wind} (m/s) • Temperature T ($°C$)	• Rotor radius R (m) • Rated power P_{rated} (W) • Cut-in, rated, and cut-out wind speeds • $C_{p,opt}$	• Electric Power (W)

- $(v_{cut-in}, v_{rated}, v_{cut-out})$
- Generator efficiency η_G

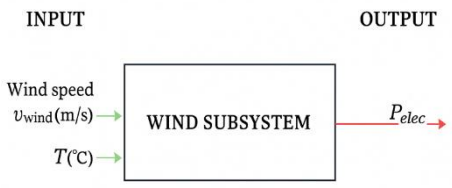

Figure 11. Block diagram for the wind turbine.

2.3.2 OpenModelica Implementation

The wind turbine was modeled in OpenModelica using steady-state aerodynamic equations and tested under fixed wind conditions to verify rated power output.

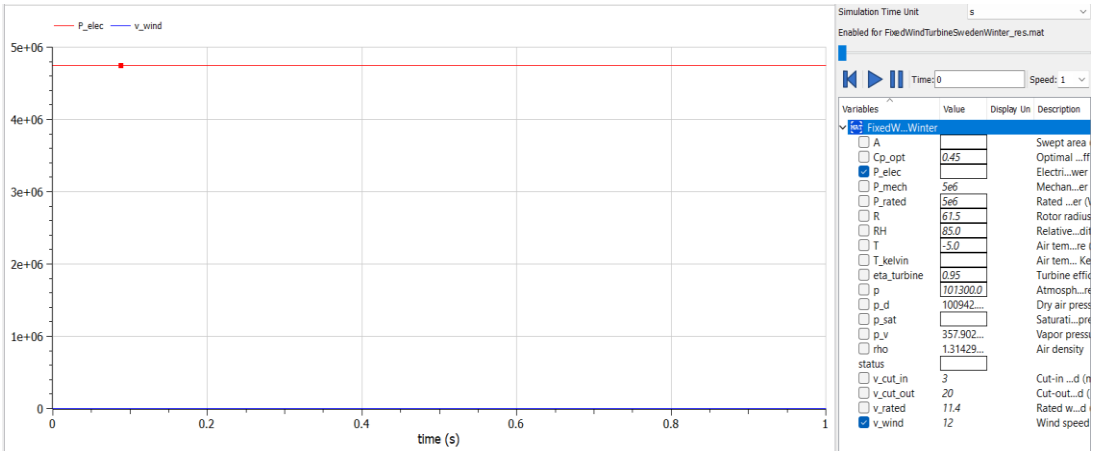

Figure 12. Simulation Wind power in OpenModelica

2.4 Battery Model

In this system, battery energy storage plays a critical role in maintaining power stability and ensuring a continuous supply of power. The battery model described in this document is developed using Modelica and is intended for use in hybrid energy systems, where it acts as a buffer supplying additional power during demand peaks or generation shortfalls. This model is designed for simulating energy flow and state-of-charge (SOC) dynamics. It considers charge/discharge limits and enforces minimum and maximum SOC constraints, ensuring realistic and safe battery operation.

2.4.1 SOC Dynamics

The rate of change of SOC depends on the power input P_bat and the total energy capacity E_max. However, charging or discharging is blocked if the battery is at its minimum or maximum SOC:

$$\frac{dSOC}{dt} = \{if\,(SOC \leq SOC\_min\ then\ P\_bat < 0.2)\ or\ (SOC \geq SOC\_min\ then\ P\_bat > 0.2)\ else\ \}$$

$$\frac{dSOC}{dt} = \frac{P\_bat}{E\_max}$$

This equation controls the rate of change of the battery's state of charge (SOC) based on its current limits and the requested charging or discharging power. The table below provides a structured summary of the battery model implemented in Modelica. It captures the essential behavior of the battery based on energy flow, state of charge (SOC), and operational constraints.

Table 7. I/O and Parameters for the battery model.

CATEGORY	NAME	UNIT	DESCRIPTION
Input	Real P_bat	W	Request power to or from the battery
Output	Real SOC	[0.2-0.98]	State of charge, initial value 0.5
Output	Real P_available	W	Power available for discharge
Parameter	E_max	MWH	Maximum energy capacity (1.4MWH)
Parameter	P_max	W	Maximum discharge or charge power (400k)
Internal variable	E_available	WH	Energy available for discharge

Figure 13. Block diagram of the battery model.

```
def test_battery():
    print("\n=== BatteryModel self-test ===")
    B = myFMU(batfmu)

    # Start at SOC = 0.50
    B.init(0.0, [('SOC', 0.50)])

    # Step 1 : charge at +100 kW for 60 s
    B.set('P_bat', 100000.0)     # + charge (W)
    B.doStep(0.0, 3600.0)
    soc1 = B.get('SOC')
    print("After +100 kW × 1hr  → SOC =", soc1)

    # Step 2 : discharge at -150 kW for 60 s
    B.set('P_bat', -150000.0)    # - discharge (W)
    B.doStep(60.0, 3*3600.0)
    soc2 = B.get('SOC')
    print("After -150 kW × 3 hr → SOC =", soc2)

    B.terminate()
```

Figure 14. Unit test for the battery in Python.

In Figure 17, a test unit in Python for the battery model was implemented using the FMU file generated from OpenModelica. When run, it effectively managed to charge or discharge the battery and change the SOC.

2.5 Heater

Heater models convert electrical power into thermal power to raise the temperature of a mass flow of water to a defined set-point. It sits downstream of renewable or dispatchable generators, absorbing excess electrical energy and delivering heat to the network.

The heater subsystem uses electrical energy to heat a fluid (water) from its inlet temperature up to a controlled setpoint. In a microgrid context, it can act as a flexible thermal load, consuming surplus renewable power (e.g., PV or wind) or grid electricity when prices are low. Its main objectives are:

- Thermal supply: Provide a controllable heat output for district or process heating.
- Grid flexibility: Serve as an electrical load that can be curtailed or ramped up to help balance supply and demand.

Table 8. Key Parameters and I/O for Heater Model.

Parameter	***Symbol***	***Value***	***Unit***	***Description***
Specific heat of water	cp_water	4186	J/(kg·K)	Thermal capacity per kg of water
Maximum electrical power	P_max	11×10^6	W	Upper limit on electrical draw
Heater efficiency	eta	0.95	–	Fraction of electrical power converted
Target temperature	T_setpoint	363.5	K	Desired outlet water temperature
Port	**Variable**	**Type**	**Unit**	**Description**
Inlet mass flow	m_flow	input Real	kg/s	Mass flow rate of water into heater
Inlet temperature	T_in	input Real	K	Temperature of incoming water
Inlet heat from gas turbine	Q_heat	input Real	W	Heat recovered from the gas turbine
Temperature achieved	T_out	output Real	K	Temperature achieved in the heater
Required Electrical Power	P_req	output Real	W	Power needed to raise water to set-point
Heat transferred to the water	Q_transferred	output Real	W	Total energy transferred to the water to heat it from both sources (Q_heat and P_req)

The heater's behavior is governed by energy balance and efficiency:

$$P_{req} = mflow \times cp_{water} \times (T_{setpoint} - T_{in})$$

$$P_{eleectrical} = \frac{Preq}{\eta}$$

Preq is the thermal power required to heat the mass flow from the actual temperature to the setpoint and P Pelectrical is the electrical power input, accounting for heater efficiency.

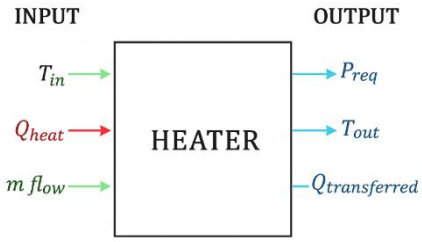

Figure 15. Block diagram of the model.

2.5.1 Test Unit

The heater test unit is introduced in the report as a concise, reproducible validation routine for the Heater FMU. It is presented immediately after the model description to demonstrate the end-to-end workflow, from instantiation of the FMU wrapper and initialization at time zero, through setting of inlet temperature and mass flow inputs, to advancing the simulation and retrieving the key outputs. By grouping these steps into a single test_heater() function. This test unit highlights the correctness of the heater implementation. As such, the test_heater() routine functions both as a validation check.

```python
def test_heater():
    print("\n=== Heater self-test ===")
    H = myFMU(heaterfmu)

    H.init(0.0, [])

    # Example: 1 kg s-1 of water, 300 K inlet (≈27 °C)
    H.set(['T_in', 'm_flow'], [300.0, 1.0])
    H.doStep(0.0, 60.0)

    print("P_req        [W] :", H.get('P_req'))
    print("P_electrical [W] :", H.get('P_electrical'))

    H.terminate()
```

```
model testHeater
  // Test inputs
  parameter Real m_flow = 1       "kg/s";
  parameter Real T_in   = 210     "K";
  parameter Real Q_heat = 150000;
  // Instantiate your new Heater
  Heater heater(
    m_flow = m_flow,
    T_in   = T_in,
    Q_heat = Q_heat);

  // Outputs to inspect
  output Real P_req        = heater.P_req;
  output Real T_out  = heater.T_out;
  output Real Q_provided  = heater.Q_provided;
  annotation( ...);
end testHeater;
```

Figure 16. Unit test for the heater in Python and OpenModelica.

In Figure 19, the results of the simulation of the model with a test unit are shown for a mass flowrate of 1kg/s, a setpoint of 363.15 K and an inlet temperature of the water of 215 K.

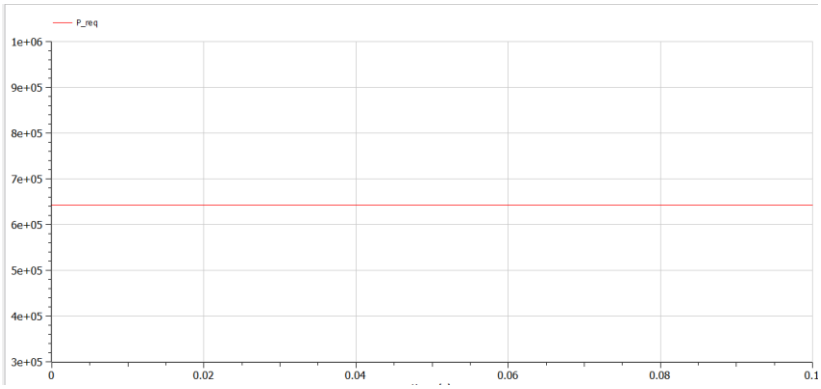

Figure 17. Result of the test unit in OpenModelica.

2.6 Controller

The supervisory controller (lo) orchestrates the flow of power between renewables, the battery, the gas turbine, and the grid export to meet the heater's thermal demand while respecting SOC and generation limits. It sits at the top level of the co-simulation, reading instantaneous PV and wind outputs, battery SOC, and the heater's power requirement, then issuing set-points for battery charge/discharge, CC output, and export.

The controller's primary objective is to ensure that the heater's power demand is met at each step by intelligently balancing the available renewable energy, the battery's charge and discharge capabilities, and the gas turbine's operating limits, while also managing any surplus through export to the grid. At the start of each simulation step, the controller measures the instantaneous output of PV and wind, summing these to form the total renewable supply. It then compares this supply directly against the heater's requirement: if renewables alone suffice, the excess energy is first used to charge the battery and any remaining surplus is routed to the grid. Conversely, when renewables fall short, the controller taps the battery down to its minimum allowable state of charge before bringing the gas turbine online, sizing its output to cover the remaining deficit while respecting the turbine's minimum and maximum loading constraints. Any extra capacity provided by the turbine above what the heater needs is again sent to the grid, ensuring no available power is wasted.

Embedded within this decision sequence is a safeguard on the battery's discharge capability, which is dynamically calculated based on its current state of charge and energy capacity and capped at the battery's maximum power rating. This prevents over-discharge and maintains long-term battery health. Similarly, the gas turbine dispatch is bounded by a lower limit that prevents inefficient low-load operation and an upper limit that protects the machinery from overload. Through this layered hierarchy the controller achieves its goal of cost-effective, reliable, and constraint-aware operation, seamlessly integrating multiple energy assets to satisfy thermal demand while optimizing the use of clean energy and preserving system integrity.

Table 9. Parameters for the controller.

Parameter	***Symbol***	***Value***	***Unit***	***Description***
Max battery SOC	SOC_max	0.98	–	Upper allowable state-of-charge
Min battery SOC	SOC_min	0.20	–	Lower allowable state-of-charge
CC minimum power	P_CC_min	350 000	W	Minimum gas turbine output
CC maximum power	P_CC_max	1 000 000	W	Maximum gas turbine output
Battery energy capacity	E_max	$1.4 \times 10^6 \times 3600$	J	Total stored energy at full charge
Time step	dt	900	s	Dispatch algorithm interval

Table 10. I/O of the controller.

Port	***Variable***	***Type***	***Unit***	***Description***
PV generation	P_pv	input Real	W	Instantaneous PV power
Wind generation	P_wind	input Real	W	Instantaneous wind power
Battery SOC	SOC_bat	input Real	–	Current battery state-of-charge
Battery discharge limit	P_available_bat	input Real	W	Max battery discharge power available this step
Heater demand	P_req	input Real	W	Thermal power requested by Heater
Battery power output	P_bat	output Real	W	Positive → charging, negative → discharging
Gas turbine set-point	P_CCreq	output Real	W	Power demanded from the gas turbine
Grid export	P_export	output Real	W	Excess power sent to the grid

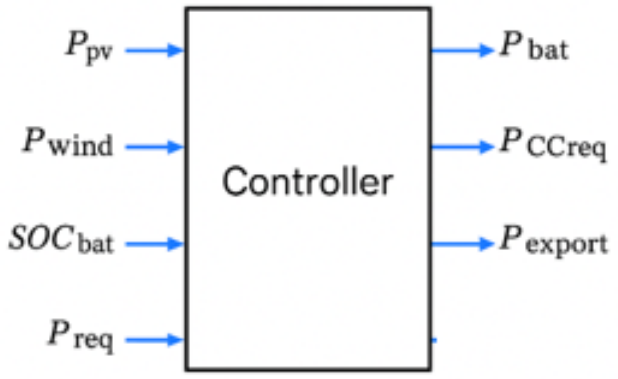

Figure 18. Block diagram for the controller

2.6.1 Test Unit

The test_controller() routine provides a quick, end-to-end check of your supervisory controller FMU to confirm it behaves as expected under a representative operating point. It wraps the compiled *controllerfmu* in your myFMU helper, initializes the simulation at time zero, then applies a sample set of inputs—200 kW PV output, 50 kW wind, 300 kW heater demand, and 60% battery SOC. After advancing the model by 60 seconds, it retrieves and prints the gas turbine set-point

(P_CCreq), battery charge/discharge command (P_bat), and any exported power (P_export). A successful run with plausible numerical results demonstrates that the control logic correctly balances renewable supply, storage, and dispatchable generation.

```python
def test_controller():
    print("\n=== Controller (lo) self-test ===")
    C = myFMU(controllerfmu)
    C.init(0.0, [])                    # t = 0 s
    # Example inputs (all in kW)
    C.set(['P_pv', 'P_req', 'P_wind', 'SOC_bat'],
          [200.0,   300.0,   50.0,      0.60])
    C.doStep(0.0, 60.0)                # 60 s step
    print("P_CCreq  [kW] :", C.get('P_CCreq'))
    print("P_bat    [kW] :", C.get('P_bat'))
    print("P_export [kW] :", C.get('P_export'))

    C.terminate()
```

```
model TestHeaterDispatch
    Real P_wind = 250e3;
    Real T_water_test = 250;
    Real m_flow = 1.0; // for example, in kg/s
    Real Irr =  500e3 * (1 + 0.2 * sin(2 * Modelica.Constants.pi * time));
    Real T_amb = 301;

    // Instantiate components
    Heater        h;
    lo            disp;
    BatteryModel  bat;
    PV_model pva;
    GasTurbine GT;


  equation
    // Dispatch inputs ( and SOC)

    GT.T_amb = T_amb;
    GT.P_CCreq = disp.P_CCreq;
    pva.Irr = Irr;
    pva.Temp = T_amb - 273.15;
    h.m_flow = 1.0;
    h.T_in = T_water_test;
    disp.P_pv = pva.P_pv;
    disp.P_wind          = P_wind;
    disp.P_req           = h.P_req;
    disp.SOC_bat         = bat.SOC;
    bat.P_bat = disp.P_bat;

annotation (experiment( ...);
end TestHeaterDispatch;
```

Figure 19. Test unit for the controller in Python and OpenModelica.

3 Results

This section presents the results of the simulation, which aimed to maintain a constant outlet water temperature of 363.5 K over a 72-hour period (sunny, rainy, and snowy days), using a hybrid energy system. The figures that follows demonstrate how the system balances power supply from PV, wind, battery, and gas turbine sources while responding to changing weather conditions and heater demand. Key aspects such as power generation, energy export, battery behavior, constraint adherence, and temperature regulation are analyzed to assess whether the controller meets performance targets within the co-simulation framework used to couple the thermal and electrical subsystems.

3.1 Simulation Results

1. Renewable Power and Demand Balancing:

Figure 20 shows the electrical power contributions from PV, wind, and the gas turbine, along with the heater's power demand (P_req) and the power exported to the grid (P_export). Throughout the 72-hour period, P_req remains steady at 400 kW with a slight decrease when the turbine is turn on due to the heat recovered that decreases the electric demand of the heater. PV and wind generation vary significantly depending on environmental conditions. The gas turbine activates when renewables are insufficient, with output rising above 350 kW only during power shortfalls. Exported power appears only when both demand and battery storage have been satisfied, confirming that surplus energy is not wasted.

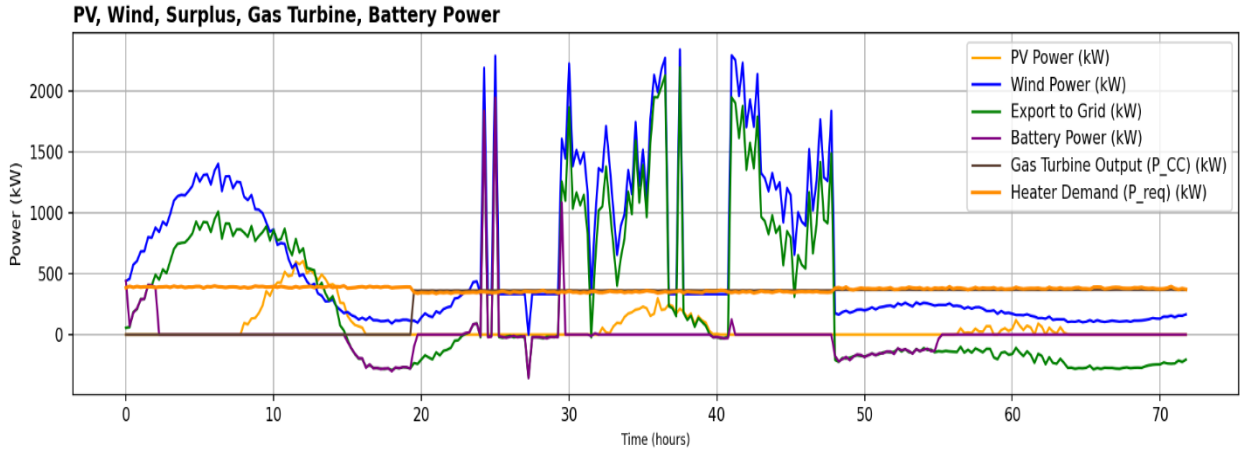

Figure 20. Power generation from PV, wind, and gas turbine compared to heater demand and exported power.

2. Heater Energy Balance

The figure 21 compares the power input to the heater (P_req), the heat recovered from the gas turbine (Q_heat), and the total heat transferred to the water (Q_transferred). The plot shows that Q_transferred consistently exceeds P_req, indicating that the gas turbine's recovered heat (Q_heat) supplements the electrical heating. The recovered heat reaches a steady value of around 45 kW

when active. This confirms the efficiency of the hybrid heating strategy, which reduces reliance on electrical input by leveraging recovered energy.

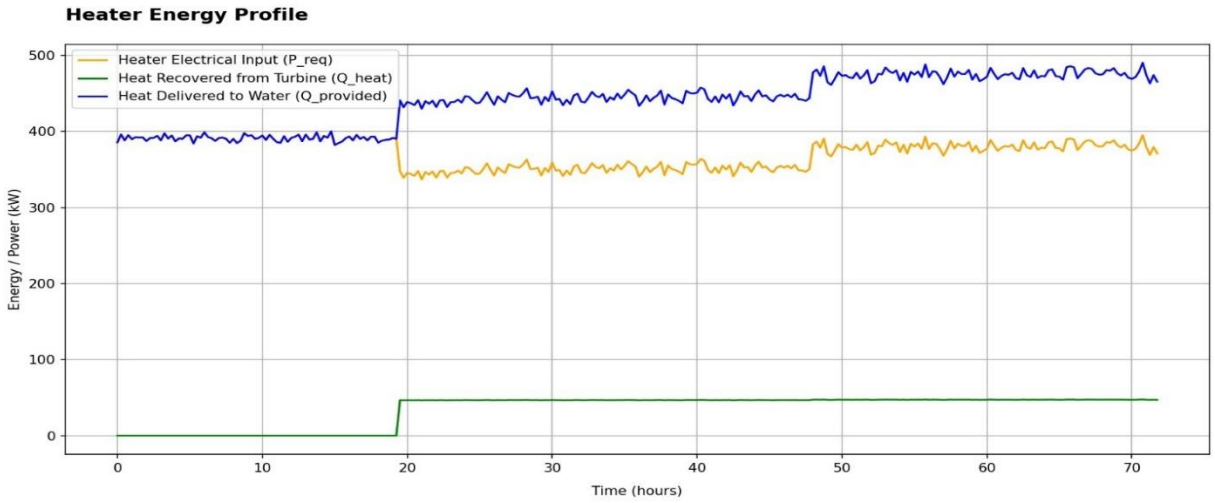

Figure 21. Heater demand met using electrical input and recovered gas turbine heat.

3. Constraint Adherence Checks

The figure 22 confirms that the controller respects all key system constraints. The SOC remains within the safe range of 0.2 to 0.98, gas turbine requests (P_CCreq) stay between 350 kW and 1000 kW, and battery power (P_bat) remains below the ±400 kW limit. The battery responds to renewable surplus and deficit conditions without exceeding any bounds, demonstrating reliable and constraint-compliant energy management.

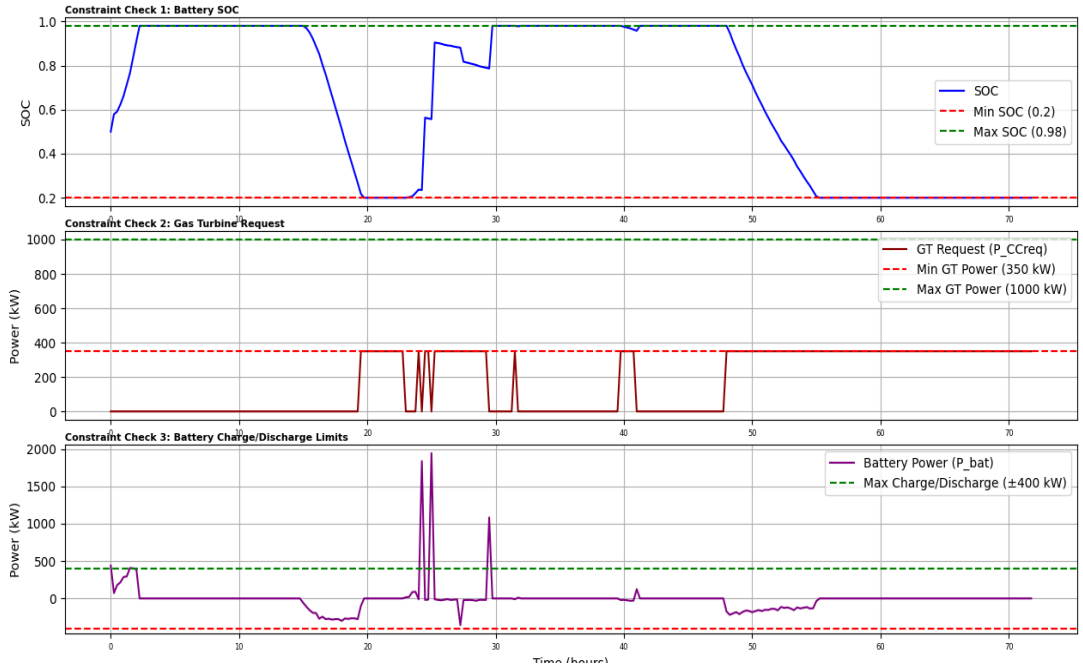

Figure 22. SOC, gas turbine request, and battery power remain within operational constraints.

4. Heater Inlet and Outlet Temperatures

The figure 23 highlights the temperature regulation performance of the system. The inlet water temperature (T_in) fluctuates based on environmental conditions, ranging roughly from 250 K to 280 K. In contrast, the outlet temperature (T_out) remains stable at approximately 363.5 K for the entire duration of the simulation. The outlet temperature stays within the defined acceptable range of 358.5 K to 368.5 K, demonstrating that the control system successfully maintains the desired thermal output.

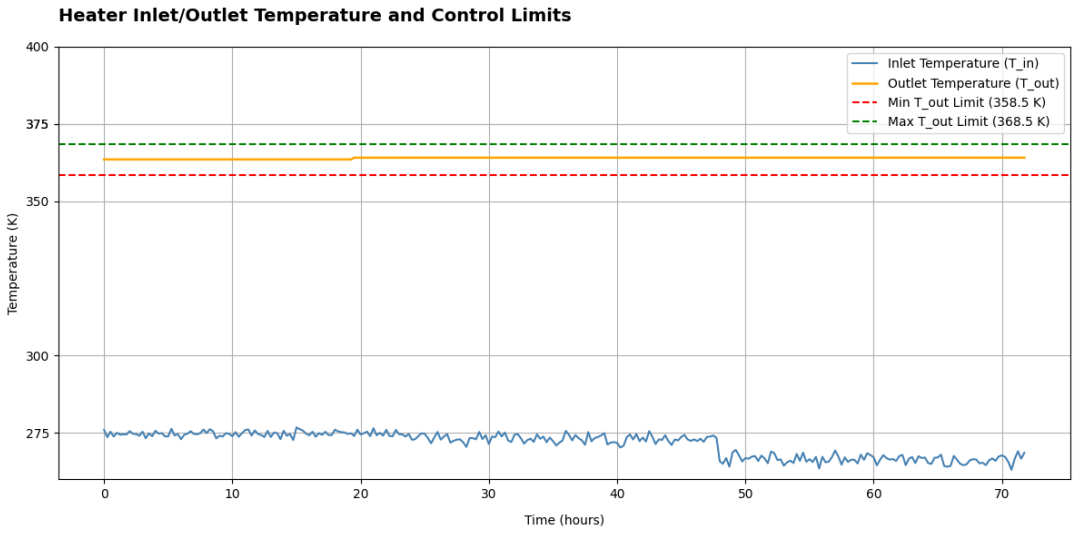

Figure 23. Heater demand met using electrical input and recovered gas turbine heat.

5. Grid Export and Battery SOC

The 24 figure shows that power is only exported to the grid when the battery is nearly full, typically at an SOC of around 0.98. This behavior confirms that the controller prioritizes using renewable energy to meet internal demand and charge the battery before allowing export, reflecting efficient and intentional energy management.

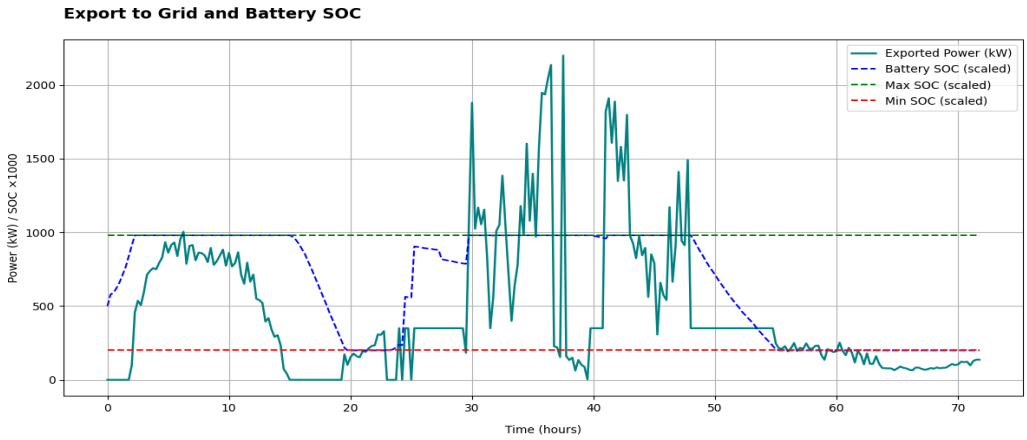

Figure 24. Grid export permitted only when battery SOC approaches upper limit.

4 Conclusion

This project demonstrates the value of co-simulation for assessing a hybrid renewable energy system that must deliver constant thermal output in a district-heating context. By coordinating photovoltaic, wind, battery, and gas-turbine resources, the controller kept the outlet-water temperature at 363.5 K for 72 h, despite weather variability, while honoring all operational limits on SOC, turbine power, and battery charge rates.

Results show that the controller balances supply and demand, prioritizes renewable input, and exports electricity only when storage is full; each subsystem staying within its physical bounds. The plots confirm seamless interaction between thermal and electrical domains inside the co-simulation framework.

Pedagogically, the work aligns with course themes of DEVS formalism, modular modelling, and heterogeneous co-simulation. Continuous-time components (heater, power electronics) interact in real time with event-driven control and battery logic through standardized interfaces, illustrating the power of modular hybrid simulation.

In conclusion, the study validates both the controller's effectiveness and co-simulation's strength as a tool for evaluating complex, multi-domain energy systems, offering a scalable approach for future smart-grid and district-heating applications.

4.1 Advantages of this System

This co-simulated hybrid energy system offers several key advantages, especially in the context of cold-climate residential heating and renewable energy integration. It combines modularity, local control, thermal-electric efficiency, and flexibility for future expansion.

- **No External Communication Required**: The embedded controller autonomously manages power flows using local measurements from each FMU, reducing complexity and improving reliability in isolated or harsh environments.
- **FMU-Based Modularity**: Each subsystem (PV, wind, gas turbine, battery, heater, and controller) was modeled and tested individually in OpenModelica and integrated via Python, making the system reusable and scalable.
- **Cold Climate Optimization**: The system is designed to ensure reliable heating during Swedish winters, activating the gas turbine only when renewable sources cannot meet demand.
- **Combined Heat and Power**: The gas turbine produces both electricity and thermal energy, with heat recovery directed to a district heating network to maximize efficiency.
- **Expandable Architecture**: New FMUs, such as EV chargers or advanced control modules, can be added without reworking the entire system, thanks to the modular Python-based orchestration.
- **Combined Heat and Power (CHP):** The gas turbine produces both electricity and heat. Recovered heat is fed into the district heating network, significantly improving overall system efficiency.

5 Future Improvements

An enhanced gas turbine model could feature a closed-loop fuel control driven by the central controller, which dynamically adjusts fuel flow to meet power demand while compensating for ambient temperature changes and now include an exhaust temperature output that feeds back into the controller to monitor performance and enable heat recovery in combined heat-and-power applications.

Future PV modeling will replace the fixed-efficiency calculation with a temperature-dependent cell model using I–V curves, integrate maximum-power-point tracking logic to optimize output under varying irradiance, and incorporate inverter and power-electronics dynamics to capture switching losses and more accurately reflect real-world system response.

To improve realism, the battery model will simulate capacity fading over cycles to represent long-term degradation and include temperature dependence of both capacity and efficiency so that performance varies with ambient conditions.

Advancements in the wind turbine model will add robust pitch, yaw, and torque control loops that enable the system to respond to changing wind conditions, couple aerodynamic torque with generator dynamics for realistic transient behavior and address current simulation instabilities by refining solver settings and stability constraints.

The heater model will gain thermal dynamics by introducing warm-up inertia and time constants, integrate embedded PID or on/off control logic with enforced maximum power limits, and adopt variable efficiency and capacity as functions of load and temperature, along with safety features such as over-temperature and boil-protection limits.

Improvements to the central controller include support for sub-hourly timesteps with dynamic state-of-charge tracking, enforcement of turbine ramp-rate and minimum-run-time constraints as well as battery degradation limits, and the addition of an optimization layer to minimize cost or CO_2 emissions while managing grid import/export and renewable curtailment constraints.

6 Appendix

Python Code for the Co-Simulation

```
from myFMU.myFMU import myFMU
import pandas as pd

fmuDirw="FMUDir/" # windows architecture
rep="systemproject/"
path= fmuDirw+rep

controllerfile="lo.fmu"
heaterfile="Heater.fmu"
windfile = "WindT.fmu"
pvfile ="PV_model.fmu"
gastfile ="GasTurbine.fmu"
batfile= "BatteryModel.fmu"

controllerfmu=path+controllerfile
heaterfmu=path+heaterfile
gastfmu=path+gastfile
pvfmu=path+pvfile
batfmu=path+batfile
windfmu  = path + windfile

import numpy as np

def simulate_energy_system(csv_path, mflow_const=1.0):
    # Load and clean column names
    data = pd.read_csv(csv_path)
    data.columns = data.columns.str.strip()
    data.rename(columns={
        'Irr':    'Irr',
        'S_wind': 'v_wind',
        'Tamb':   'T_amb',
        'Twater': 'T_water'
    }, inplace=True)

    # Extract series
    n = len(data)
    irr_series    = data['Irr'].values
    vwind_series  = data['v_wind'].values
    tamb_series   = data['T_amb'].values
    twater_series = data['T_water'].values
    mflow_series  = np.full(n, mflow_const)

    # Instantiate FMUs
    Fcontroller = myFMU(controllerfmu)
    Fbat        = myFMU(batfmu)
    Fheater     = myFMU(heaterfmu)
    Fgast       = myFMU(gastfmu)
    Fpv         = myFMU(pvfmu)
    Fwind       = myFMU(windfmu)

    # Initialize simulation
    t      = 0.0
    deltaT = 900.0

    Fcontroller.init(t, [])
    Fbat.init(t, [('SOC', 0.5), ('P_bat', 0.0)])
```

```
Fheater.init(t, [])
Fgast.init(t, [])
Fpv.init(t, [('n_pv', 10000)])
Fwind.init(t, [('P_rated', 350000)])
# Prepare results container
results = {key: [] for key in (
    'time', 'Irr', 'v_wind', 'T_amb', 'T_water',
    'P_elec_wind',
    'P_pv',
    'P_req', 'Q_provided', 'T_in', 'T_out',
    'SOC', 'P_CCreq', 'P_bat', 'P_export',
    'P_CC', 'Q_heat'
)}

# Simulation loop
q_heat = 0.0
    # --- Initialize loop variables so they exist on iter=0 ---  ## <<< init
p_req = 0.0                                    ## <<< init
p_bat = 0.0                                    ## <<< init
soc   = 0.5
for i in range(n):
    irr   = irr_series[i]
    vwind = vwind_series[i]
    Ta    = tamb_series[i]
    Tw    = twater_series[i]
    mf    = mflow_series[i]

    # Wind turbine
    Fwind.set(['T_amb','v_wind'], [Ta, vwind])
    Fwind.doStep(t, deltaT)
    p_elec_wind = Fwind.get('P_elec')

    # PV
    Fpv.set(['Irr','Temp'], [irr, Ta - 273.15])
    Fpv.doStep(t, deltaT)
    p_pv = Fpv.get('P_pv')

    # Battery update
    Fbat.set('P_bat', p_bat)
    Fbat.doStep(t, deltaT)
    soc = Fbat.get('SOC')

    ## <<< CHANGE: compute max battery discharge capacity this step (W)
    P_bat_max = 400000.0
    P_available_bat = max(0.0, (soc - 0.2) * (1.4e6 * 3600) / deltaT)
    P_available_bat = min(P_available_bat, P_bat_max)
    ## <<< END CHANGE

    # Dispatch (“lo” FMU) with all five inputs
    Fcontroller.set(
        ['P_pv','P_wind','SOC_bat','P_available_bat','P_req'],
        [p_pv, p_elec_wind, soc, P_available_bat, p_req]
    )
    Fcontroller.doStep(t, deltaT)
    p_ccreq = Fcontroller.get('P_CCreq')
    p_bat   = Fcontroller.get('P_bat')
    p_exp   = Fcontroller.get('P_export')

    # Gas turbine
    Fgast.set(['P_CCreq','T_amb'], [p_ccreq, Ta])
    Fgast.doStep(t, deltaT)
    p_cc   = Fgast.get('P_CC')
```

```
        q_heat = Fgast.get('Q_heat')

        # Heater
        Fheater.set(['T_in','m_flow','Q_heat'], [Tw, mf, q_heat])
        Fheater.doStep(t, deltaT)
        p_req      = Fheater.get('P_req')
        q_provided = Fheater.get('Q_provided')
        t_out      = Fheater.get('T_out')

        # Store results
        results['time']         .append(t)
        results['Irr']          .append(irr)
        results['v_wind']       .append(vwind)
        results['T_amb']        .append(Ta)
        results['T_water']      .append(Tw)
        results['P_elec_wind']  .append(p_elec_wind)
        results['P_pv']         .append(p_pv)
        results['P_req']        .append(p_req)
        results['Q_provided']   .append(q_provided)
        results['T_in']         .append(Tw)
        results['T_out']        .append(t_out)
        results['SOC']          .append(soc)
        results['P_CCreq']      .append(p_ccreq)
        results['P_bat']        .append(p_bat)
        results['P_export']     .append(p_exp)
        results['P_CC']         .append(p_cc)
        results['Q_heat']       .append(q_heat)

        t += deltaT

    return pd.DataFrame(results)

csv_path = "data.csv"
df = simulate_energy_system(csv_path)
df.to_csv('results.csv', index=False)   # save full results
```