me193e-final-project

May 8, 2026

1 Final Project: Data Center Coupled with Thermal Energy Storage

Spring 2026

Course: ME 193E Next-Generation Data Centers & Energy Management

Name: Austin Quan

Email: austinquan@berkeley.edu

Objective: Model a data center coupled with thermal energy storage. This will be accomplished by integrating TES into the data center model created in Project 1.

1.1 Project 1 Overview

Project 1 is a data center model used to evaluate the ensemble coefficient of performance COP_{sys} of a data center as ambient wet-bulb temperature T_{wb} changes. We accounted for the transition between economizer (free cooling) and chiller-dominated modes based on the T_{wb} and the required chilled water supply temperature. We defined realistic temperatures for the cooling tower and the heat exchangers such that heat is actually exchanged. We used our previous component models to estimate the power $\dot{W}$ for each device. For fans and pumps, we assumed power scales with the cube of the flow rate where applicable. This is however, not the most optimal method for running a data center. While there are other components that could be adjusted dynamically such as fan speed, I believe thermal energy storage is a very underrated method for doing this. When the data center is in periods of low demand or favorable ambient conditions, we can use extra power to harness "stored cooling capacity" and release it during periods of higher demand.

For Project 1, we assumed

- Medium-to-large scale enterprise data center
- Total IT load $\dot{Q}_{\text{IT}} = 10$ MW (assumed constant initially, dynamic load will be introduced)
- Raised-floor configuration with hot-aisle and cold-aisle containment
- High-performance chips cooled via heat sinks and fans
- Room level: Computer room air handlers (CRAH)
- Plant level: Water-cooled chiller plant with integrated water-side economizer
- Heat rejection: Induced-draft cooling towers
- Chilled water set point: 15°C
- CRAH return air: 35°C
- Pump/fan isentropic efficiency: 75%

1.1.1 Introduction: Data Center Basics

In order to understand where and why thermal energy storage is useful, we will first break down the basics of data centers in detail qualitatively.

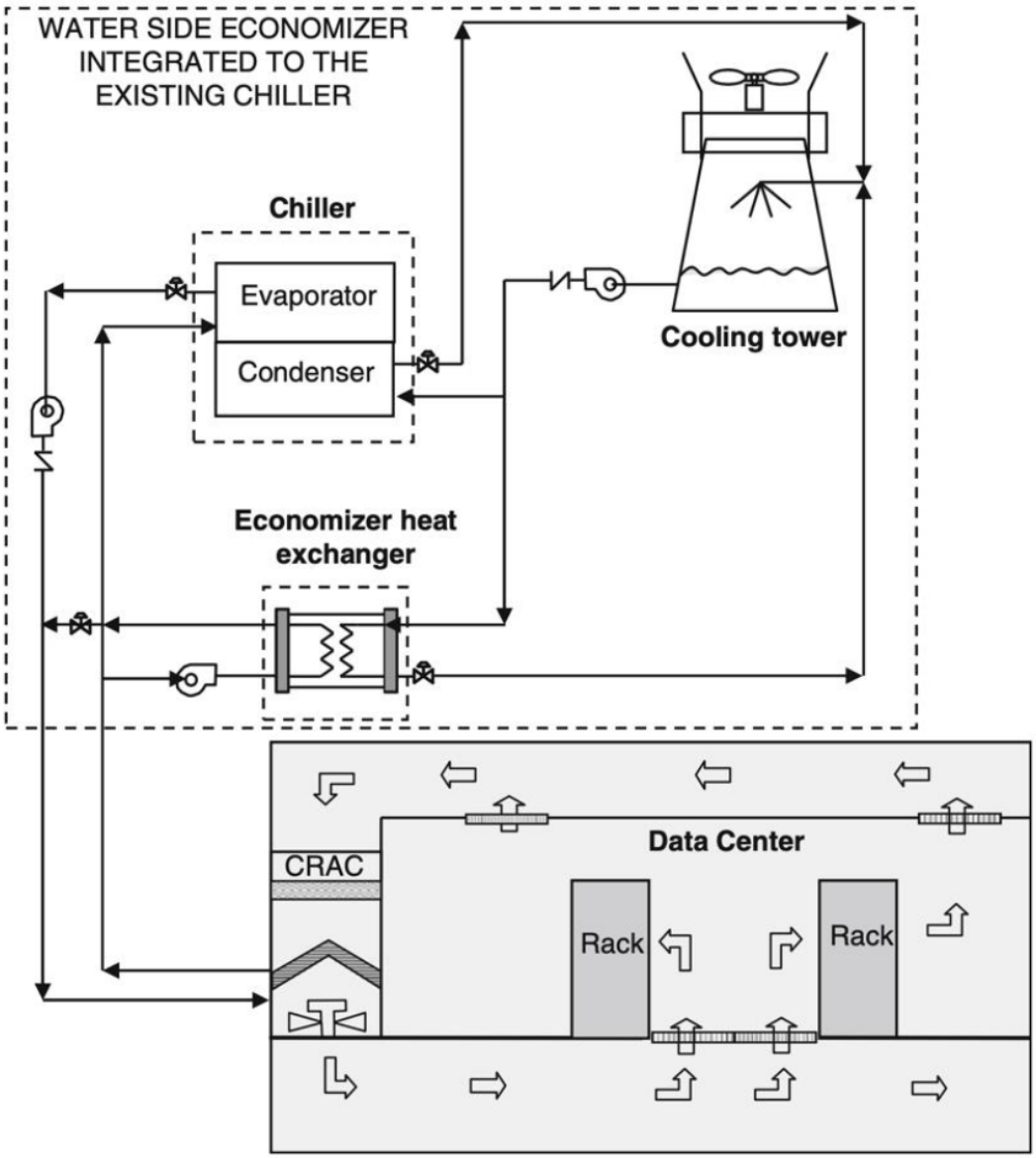

Figure 1. Y. Joshi, P. Kumar. Energy Efficient Thermal Management of Data Centers, 2012.

One of the biggest trends in engineering right now is the development of hyperscaler data centers, which are geared towards AI. As described by Jensen Huang, the current AI ecosystem is a "5-layer cake." From bottom to top, we have energy, chips, infrastructure, models, and applications. The bottom three layers are the physical hardware where this project lives. At the lowest level of all data centers, there are chips. While many data centers use CPUs, the modern trend for AI is to use

GPUs for computing. They are housed within server modules which are stacked in racks. These racks are found inside of server rooms. To keep these server rooms running, there is an electrical demand for the IT, lighting, fans, and methods to cool the chips such as direct liquid cooling. All of these components use power which in turn must leave the room as heat.

To manage this, the heat first must go through a CRAC or CRAH. The CRAC, computer room air conditioner, is used for smaller data centers whereas the CRAH, computer room air handler, is better for larger server rooms; this is mostly because they are more energy efficient and easier to scale. A CRAH essentially takes in the hot air from a room into coils which chill the air and then blow it throughout the room.

Then, the now chilled water from the CRAH leaves the room through a chilled water loop.

After that, we have an economizer and a chiller. The economizer provides free cooling, which is sufficient if conditions are favorable and demand is not too high. The economizer is a heat exchanger that circulates hot and cold water from the cooling tower. If the economizer is not sufficient, the chiller takes over. It is essentially a refrigeration system. When it takes over, it dominates the energy demand for the data center.

Lastly, we have the cooling tower in which heat is expelled into the atmosphere as steam (not fog!).

1.1.2 Governing Equations

We are given a constant heat load of $\dot{Q}_{\text{IT}} = 10\text{MW}$. We assume a wet-bulb temperature that varies from 0°C to 35°C. Using the fact that the ensemble COP is given by heat extracted divided by the net work input, our final goal is

$$\text{COP}_{\text{sys}} = \frac{\dot{Q}_{\text{IT}}}{\dot{W}_{\text{chip}} + \dot{W}_{\text{rack}} + \dot{W}_{\text{CRAC}} + \dot{W}_{\text{pumps}} + \dot{W}_{\text{chiller}} + \dot{W}_{\text{CT}}}.$$

where $\dot{Q}_{\text{IT}} \approx \dot{Q}_{\text{chiller}}$ since the chiller dominates COP.

The psychrometric principle used here begins with the inlet conditions at one state. We need pressure p, dry-bulb temperature T, and relative humidity ϕ. Then, the saturation vapor pressure at a temperature T is given by

$$p_{v,\text{sat}}(T).$$

And the partial pressure of vapor at the inlet is given by

$$p_{v,1} = \phi_1 p_{v,\text{sat}}(T_1).$$

We also have humidity ratio

$$\omega_1 = 0.622 \frac{p_{v,1}}{p - p_{v,1}}$$

and enthalpy

$$h_1 = h_a(T_1) + \omega_1 h_v(T_1).$$

Rack

We assume that all of the power from IT becomes heat that must be removed. Since $\dot{Q}_{\text{IT}}$ is fixed, we can hold the airflow through the rack as fixed and treat $\dot{W}_{\text{rack}}$ as a constant. For this layer of the data center, a thermal load enters the room and must be removed. This will be defined as

$$\dot{Q}_{\text{room}} = \dot{Q}_{\text{IT}}.$$

CRAH

To summarize the next few components, we have

- State 1: air entering the CRAH
- State 2: air exiting the CRAH
- State 3: potentially liquid water exiting the CRAH
- State 4: chilled water entering the CRAH
- State 5: chilled water exiting the CRAH
- State 6: cooling tower water entering the economizer
- State 7: cooling tower water exiting the economizer

For the CRAH, we have air entering the inlet and air exiting the outlet. We also have chilled water entering and exiting. For state 3, we have $\dot{m}_3 = \dot{m}_{\text{a}}\,(\omega_1 - \omega_2)$ and $h_3 = h_f(T_2)$. We will assume a constant mass flow rate for dry air $\dot{m}_a$.

We have the mass balance for dry air and water, respectively:

$$\dot{m}_{\text{a1}} = \dot{m}_{\text{a2}} = \dot{m}_{\text{a}}$$

$$\dot{m}_{\text{a1}}\omega_1 = \dot{m}_3 + \dot{m}_{\text{a2}}\omega_2$$

We have the energy balance, where we assume steady flow and neglect KE and PE:

$$\dot{m}_{\text{a1}} h_1 + \dot{W}_{\text{fan}} = \dot{m}_{\text{a2}} h_2 + \dot{m}_3 h_3 + \dot{Q}_{\text{out}}$$

The cooling rate (heat extracted by the coil) is

$$\dot{Q}_{\text{out}} = \dot{m}_{\text{a1}} h_1 + \dot{W}_{\text{fan}} - (\dot{m}_{\text{a2}} h_2 + \dot{m}_3 h_3)$$

where $\dot{W}_{\text{fan}} = \frac{\Delta p \dot{V}}{\eta}$.

We now can get the COP for this component:

$$\text{COP}_{\text{R}} = \frac{\dot{Q}_{\text{L}}}{\dot{W}_{\text{in,compressor}}}$$

with $\dot{Q}_{\mathrm{L}} = \dot{Q}_{\mathrm{out}}$. Thus,

$$\dot{W}_{\mathrm{in,compressor}} = \frac{\dot{Q}_{\mathrm{out}}}{\mathrm{COP}_{\mathrm{R}}}.$$

Substituting, we get

$$\dot{W}_{\mathrm{in,compressor}} = \frac{\dot{m}_{\mathrm{a}} \left[h_1 - h_2 - (\omega_1 - \omega_2)\, h_3\right] + \dot{W}_{\mathrm{fan}}}{\mathrm{COP}_{\mathrm{R}}}.$$

Chilled Water

The required cooling load given by the CRAH determines the chilled water flow rate which determines pump work:

$$\dot{W}_{\mathrm{pump}} = \frac{\Delta p_{\mathrm{cw}} \dot{V}_{\mathrm{cw}}}{\eta_{\mathrm{pump}}}$$

On the water side of the coil, we have

$$\dot{Q}_{\mathrm{coil}} = \dot{m}_{\mathrm{cw}} c_{p,w} (T_{\mathrm{cw,return}} - T_{\mathrm{cw,supply}}).$$

with $T_{\mathrm{cw,supply}} = 15^{\circ}\mathrm{C}$.

Economizer

We will treat this as an adiabatic heat exchanger. After the same energy balance as before, we get

$$\dot{Q}_{\mathrm{coil}} = \dot{m}_{\mathrm{cw}} c_{p,w} (T_{\mathrm{cw,in}} - T_{\mathrm{cw,out}}) = \dot{m}_{\mathrm{ctw}} c_{p,w} (T_{\mathrm{ctw,in}} - T_{\mathrm{ctw,out}}).$$

Alternatively, we can get the mass flow rates from enthalpy:

$$\dot{Q}_{\mathrm{coil}} = \dot{m}_{\mathrm{cw}} (h_5 - h_4) = \dot{m}_{\mathrm{ctw}} (h_7 - h_6)$$

Chiller

For the chiller, we need cooling provided to the chilled water loop $\dot{Q}_{\mathrm{chiller}}$ and electrical input $\dot{W}_{\mathrm{chiller}}$. Then the COP is given by

$$\mathrm{COP}_{\mathrm{chiller}} = \frac{\dot{Q}_{\mathrm{chiller}}}{\dot{W}_{\mathrm{chiller}}}.$$

The chiller activates when the economizer cannot meet the cooling demand and thus T_{wb} is relevant because the chiller depends on heat rejection which is dependent on the ambient temperature.

Cooling Tower

- State 1: air entering

- State 2: air exiting
- State 3: hot water enters the tower
- State 4: cold water exits the tower

We have the mass flow rate of dry air $\dot{m}_a$ and $\dot{m}_{\text{makeup}}$.

We have the dry air and water mass balances, respectively:

$$\dot{m}_{\text{a1}} = \dot{m}_{\text{a2}} = \dot{m}_{\text{a}}$$

$$\dot{m}_3 + \dot{m}_{\text{a1}}\omega_1 = \dot{m}_4 + \dot{m}_{\text{a2}}\omega_2$$

And the energy balance:

$$\dot{m}_{\text{a1}}h_1 + \dot{m}_3 h_3 = \dot{m}_{\text{a2}}h_2 + \dot{m}_4 h_4$$

$$\dot{m}_{\text{a}}(h_1 - h_2) = \dot{m}_4 h_4 - \dot{m}_3 h_3$$

Because of evaporation, we have

$$\dot{m}_{\text{makeup}} = \dot{m}_3 - \dot{m}_4 = \dot{m}_{\text{a}}(\omega_2 - \omega_1) \implies \dot{m}_4 = \dot{m}_3 - \dot{m}_{\text{makeup}}.$$

And then we rewrite it as

$$\dot{m}_{\text{a}}(h_1 - h_2) = (\dot{m}_3 - \dot{m}_{\text{makeup}})h_4 - \dot{m}_3 h_3 = \dot{m}_3 h_4 - \dot{m}_{\text{makeup}}h_4 - \dot{m}_3 h_3$$

We can then substitute in $\dot{m}_{\text{makeup}} = \dot{m}_{\text{a}}(\omega_2 - \omega_1)$:

$$\dot{m}_{\text{a}}(h_1 - h_2) = h_4\dot{m}_3 - h_4(\dot{m}_{\text{a}}(\omega_2 - \omega_1)) - \dot{m}_3 h_3$$

$$\dot{m}_{\text{a}}[(h_1 - h_2) + h_4(\omega_2 - \omega_1)] = h_4\dot{m}_3 - \dot{m}_3 h_3$$

Finally,

$$\dot{m}_{\text{a}} = \frac{h_4\dot{m}_3 - \dot{m}_3 h_3}{(h_1 - h_2) + h_4(\omega_2 - \omega_1)} = \frac{\dot{m}_3(h_3 - h_4)}{(h_2 - h_1) - h_4(\omega_2 - \omega_1)}.$$

Now, we have the results from the original model:

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

# ------------------------------
# CONSTANTS & ASSUMPTIONS
```

```
# ----------------------------

cp_water = 4186    # J/kg-K
cp_air = 1005      # J/kg-K

Q_IT = 10e6        # IT heat load (W)
T_cw_supply = 15   # chilled water supply (deg C)
T_cw_return = 25.0
T_Return = 35
Eta_pump = 0.75
Q_rated = Q_IT

dT_tower = 4.0
dT_condenser = 6.0
dT_evap = 5.0

eta_carnot = 0.40

frac_crah = 0.02
frac_pumps = 0.01
frac_rack = 0.00
W_tower_fan = frac_crah * Q_IT

T_wb_array = np.linspace(0, 35, 100)

# ----------------------------
# Rack
# ----------------------------

def rack(Q_IT):
    Q_rack = Q_IT
    W_rack = frac_rack * Q_IT
    return Q_rack, W_rack

# ----------------------------
# CRAH
# ----------------------------

def crah(Q_IT):
    Q_coil = Q_IT
    W_crah = frac_crah * Q_IT
    return Q_coil, W_crah

# ----------------------------
# Chilled water
# ----------------------------
```

```
def chilled_water(Q_coil, Q_IT):
    mdot_cw = Q_coil / (cp_water * (T_cw_return - T_cw_supply))
    W_pumps = frac_pumps * Q_IT
    return mdot_cw, W_pumps

# ----------------------------
# Economizer (heat exchanger)
# ----------------------------

def economizer(T_cw_out, Q_IT):
    if T_cw_out <= T_cw_supply:
        return Q_IT
    elif T_cw_out <= T_cw_return:
        frac = (T_cw_return - T_cw_out) / (T_cw_return - T_cw_supply)
        return float(np.clip(frac, 0.0, 1.0)) * Q_IT
    else:
        return 0.0

# ----------------------------
# Cooling tower
# ----------------------------

def tower_temperature(T_wb):
    T_cw_out = T_wb + dT_tower
    T_cond = T_cw_out + dT_condenser
    return T_cw_out, T_cond

# ----------------------------
# Chiller
# ----------------------------

def chiller(Q_chiller, T_cond):
    if Q_chiller <= 0:
        return 0.0, 0.0

    T_evap = T_cw_supply - dT_evap

    T_cond_K = T_cond + 273.15
    T_evap_K = T_evap + 273.15

    lift = max(T_cond_K - T_evap_K, 1e-6)
    COP_carnot = T_evap_K / lift
    COP_ch = eta_carnot * COP_carnot

    W_comp = Q_chiller / COP_ch
    Q_rej = Q_chiller + W_comp
    return W_comp, Q_rej
```

```
# ----------------------------
# System model
# ----------------------------

def system_model(T_wb, Q_IT):
    Q_rack, W_rack = rack(Q_IT)
    Q_coil, W_crah = crah(Q_IT)
    mdot_cw, W_pumps = chilled_water(Q_coil, Q_IT)

    T_cw_out, T_cond = tower_temperature(T_wb)
    Q_econ = economizer(T_cw_out, Q_IT)

    Q_chiller = Q_IT - Q_econ
    W_comp, Q_rej = chiller(Q_chiller, T_cond)

    W_total = W_rack + W_crah + W_pumps + W_comp

    return Q_econ, Q_chiller, W_total

# ----------------------------
# Baseline sweep
# ----------------------------

COP_sys_array = []

for T_wb in T_wb_array:
    Q_econ, Q_chiller, W_total = system_model(T_wb, Q_IT)
    COP_sys = Q_IT / W_total
    COP_sys_array.append(COP_sys)

COP_sys_array = np.array(COP_sys_array)

# ----------------------------
# Plot
# ----------------------------

plt.figure()
plt.plot(T_wb_array, COP_sys_array)
plt.figtext(
    0.5, 0.01,
    "Figure 2",
    wrap=True,
    horizontalalignment='center',
    fontsize=12,
)
plt.subplots_adjust(bottom=0.2)
```

```
plt.xlabel("wet-bulb temperature (deg C)")
plt.ylabel("system COP")
plt.title("Ensemble COP vs Wet-Bulb Temperature")
plt.grid(True)
plt.show()
```

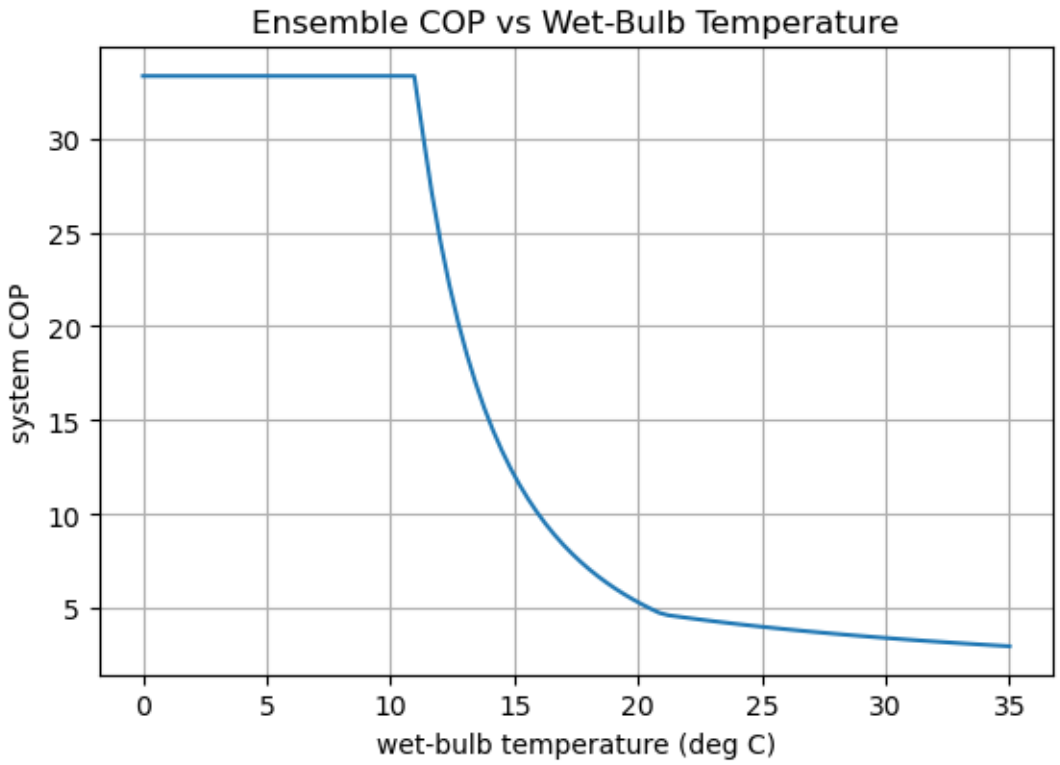

Figure 2

1.1.3 Breakpoints

The system transitions between three operating regions depending on the conditions:

1. full economizer (free cooling)
2. partial economizer
3. chiller dominated

The first breakpoint is between regions 1 and 2, which occurs when the cooling tower is no longer sufficient to meet the chilled water supply temperature:

$$T_{\text{cw,out}} = T_{\text{wb,1}} + \Delta T_{\text{tower}} = T_{\text{cw,supply}}$$

$$T_{\text{wb},1} = T_{\text{cw,supply}} - \Delta T_{\text{tower}}$$

The second breakpoint is between regions 2 and 3, which occurs when the cooling tower outlet reaches the chilled water return temperature:

$$T_{\text{cw,out}} = T_{\text{cw,return}}$$

$$T_{\text{wb},2} + \Delta T_{\text{tower}} = T_{\text{cw,return}}$$

$$T_{\text{wb},2} = T_{\text{cw,return}} - \Delta T_{\text{tower}}$$

[2]:
```
# breakpoints

T_wb_1 = T_cw_supply - dT_tower
T_wb_2 = T_cw_return - dT_tower

print(f"First breakpoint (full to partial economizer): {T_wb_1:.2f} deg C")
print(f"Second breakpoint (partial economizer to chiller-dominated): {T_wb_2:.
  ↪2f} deg C")
```

```
First breakpoint (full to partial economizer): 11.00 deg C
Second breakpoint (partial economizer to chiller-dominated): 21.00 deg C
```

1.2 Thermal Energy Storage Model 1

Thermal energy storage (TES) is the use of a thermal reservoir to store heating or cooling capacity and utilizing it when appropriate. This is becoming increasingly useful in modern energy systems where variability in thermal load and ambient conditions create inefficiencies for a static system. There are three types of TES: sensible heat storage, latent heat storage, and thermochemical storage. Sensible heat storage relies on a temperature difference within the medium to store energy; latent heat storage relies on phase-change materials; thermochemical storage relies on reversible chemical reactions. For our purposes, we will be modeling the first. This is largely because sensible heat storage is already at relatively high technological maturity, is inexpensive, and easy to implement. However, it does have some limitations such as lower energy density which means we will need a larger volume and heat losses which require insulation.

1.2.1 Infrastructure

We will place an insulated storage tank close to the chiller, connected in parallel with the chilled-water loop. This tank will be filled with a fluid that can be pumped and does not evaporate at 90 °C. A control valve and heat exchanger regulate when thermal energy is stored or released. Thus our system is cold storage.

1.2.2 Governing Equations

The total stored energy can be expressed as

$$E_{\text{sensible}} = \rho V \int_{T_c}^{T_h} c_p(T)\, dT.$$

where ρ is fluid density, V is volume, c_p is specific heat capacity, T_c is the temperature of the cold reservoir, and T_h is the temperature of the hot reservoir.

For most liquids over a small temperature range, we can assume $c_p(T) \approx c_p$, which is constant. Then, we can simplify:

$$E_{\text{sensible}} = \rho V c_p \int_{T_c}^{T_h} dT$$

$$= \rho V c_p (T_h - T_c)$$

$$= \rho V c_p \Delta T$$

In data centers, TES acts as a buffer to smooth fluctuations in cooling demand. TES on its own is not enough to provide cooling and must be used in combination with our existing cooling infrastructure. The annual energy usage of data centers is nowhere as large of a problem as peak demand. Peak demand is a massive issue because data centers are built and ran to manage peak loads; it dictates the infrastructure. It creates congestion and is difficult for local grids to support.

We will now update our thermal load:

$$\dot{Q}_{\text{IT}} = \dot{Q}_{\text{economizer}} + \dot{Q}_{\text{chiller}} + \dot{Q}_{\text{TES}}$$

This thermal load also needs to be dynamic:

$$\frac{dE_{\text{TES}}}{dt} = \dot{Q}_{\text{charge}} - \dot{Q}_{\text{discharge}}$$

$$\dot{Q}_{\text{TES}} = \dot{Q}_{\text{discharge}} - \dot{Q}_{\text{charge}}$$

where $\dot{Q}_{\text{charge}}$ is the rate at which cooling is stored and $\dot{Q}_{\text{discharge}}$ is the rate at which cooling is released.

TES operates in three modes:

$$\dot{Q}_{\text{TES}} = \begin{cases} < 0, & \text{Charging} \\ > 0, & \text{Discharging} \\ = 0, & \text{Idle} \end{cases}$$

For our system, we have to cap energy capacity and set a temperature limit so that the tank cannot be overcharged.

$$0 \leq E_{\text{TES}} \leq E_{\max}$$

$$T_{\min} \leq T_{\text{TES}} \leq T_{\max}$$

Using our previously discovered breakpoints $T_{\text{wb},1}$ and $T_{\text{wb},2}$, we will define the controls as such:

If $T_{\text{wb}} < T_{\text{wb},1}$, we charge TES. This is when we are getting free cooling from the economizer and thus have room to use surplus power to cool the fluid.

If $T_{\text{wb}} > T_{\text{wb},2}$, we discharge TES. This is when the chiller dominates the energy consumption of the data center. This is when TES can reduce the peak demand.

We will now introduce time-dependent inputs. We have the ambient condition $T_{\text{wb}}(t)$ and the thermal load $\dot{Q}_{\text{IT}}(t)$.

For dynamic ambient condition, we will just use a simple sinusoidal model:

$$T_{\text{wb}}(t) = \bar{T} + A_{\text{wb}} \sin\left(\frac{2\pi}{24}(t - \phi)\right)$$

For dynamic thermal load, we will use a simple model that introduces a simple peak:

$$\dot{Q}_{\text{IT}}(t) = \dot{Q}_{\text{base}} + A_{\text{IT}} \cos\left(\frac{2\pi}{24}(t - \phi)\right) \text{ MW}$$

We will use our old value of

$$\dot{Q}_{\text{base}} = 10$$

for the baseline thermal load. The value of amplitude will determine how much the load changes by. I have decided to go with

$$A_{\text{IT}} = \frac{\dot{Q}_{\max} - \dot{Q}_{\min}}{2} = \frac{(12 - 8)\,\text{MW}}{2} = 2$$

which is a reasonable value that provides enough variability to demonstrate the TES charging and discharging. Lastly, the peak time of thermal load will be set by the hottest time of the day, which we will assume is around 4:00 pm. Thus

$$\phi = 16$$

for the 16th hour of the day.

Our COP becomes

$$\mathrm{COP}_{\mathrm{sys}}(t) = \frac{\dot{Q}_{\mathrm{IT}}(t)}{\dot{W}_{\mathrm{rack}}(t) + \dot{W}_{\mathrm{CRAH}}(t) + \dot{W}_{\mathrm{pumps}}(t) + \dot{W}_{\mathrm{chiller}}(t)}$$

Let's discuss how this will be implemented numerically. For time stepping, we will step over a 24 hour period:

$$t = 0, \Delta t, 2\Delta t, \ldots, t_f$$

and update with

$$E_{\mathrm{TES}}^{t+1} = E_{\mathrm{TES}}^{t} + \left(\dot{Q}_{\mathrm{charge}} - \dot{Q}_{\mathrm{discharge}}\right)\Delta t$$

.

We will have an initial condition that assumes the tank is half full:

$$E_{\mathrm{TES}}^{0} = 0.5 E_{\max}$$

So for every timestep, we will

1. Evaluate $T_{\mathrm{wb}}(t)$
2. Evaluate $\dot{Q}_{\mathrm{IT}}(t)$
3. Determine economizer vs chiller contribution
4. Determine which mode we are in and update E_{TES}
5. Compute and store E_{TES}^{t} and $\dot{Q}_{\mathrm{TES}}^{t}$

[3]:
```
# ----------------------------
# TES Model 1
# ----------------------------

# Time setup
dt = 0.1                        # hours
dt_sec = dt * 3600.0
t_array = np.arange(0, 24 + dt, dt)

# Dynamic ambient condition
T_bar = 16.0                    # deg C
A_wb = 6.0                      # deg C amplitude
phi_wb = 10.0                   # peak at 4 pm

# TES parameters
E_max = 2.0e11                  # J
E_TES = 0.5 * E_max             # start half full
Q_TES_max = 2.0e6               # W

# Breakpoints
T_wb_1 = T_cw_supply - dT_tower
```

```
T_wb_2 = T_cw_return - dT_tower

# Storage arrays
T_wb_hist = []
Q_IT_hist = []
Q_econ_hist = []
Q_chiller_hist = []
Q_chiller_eff_hist = []
Q_TES_hist = []
E_TES_hist = []
W_total_base_hist = []
W_total_hist = []
COP_base_hist = []
COP_TES_hist = []

for t in t_array:

    # Dynamic wet-bulb temperature
    T_wb = T_bar + A_wb * np.sin(2 * np.pi * (t - phi_wb) / 24)

    # Dynamic IT thermal load (peak at 4 PM)
    Q_IT_t = 10e6 + 2e6 * np.cos(2 * np.pi * (t - 16) / 24)

    # Run baseline system at this timestep
    Q_econ, Q_chiller_base, W_total_base = system_model(T_wb, Q_IT_t)

    # TES control logic
    if T_wb < T_wb_1 and E_TES < E_max:
        Q_TES = -min(Q_TES_max, (E_max - E_TES) / dt_sec)
    elif T_wb > T_wb_2 and E_TES > 0:
        Q_TES = min(Q_TES_max, E_TES / dt_sec)
    else:
        Q_TES = 0.0

    # Update TES energy
    E_TES = E_TES + (-Q_TES) * dt_sec

    # Effective chiller load with TES
    Q_chiller_eff = max(Q_IT_t - Q_econ - Q_TES, 0.0)

    # Recompute chiller work using effective load
    T_cw_out, T_cond = tower_temperature(T_wb)
    W_comp_eff, Q_rej_eff = chiller(Q_chiller_eff, T_cond)

    # Recompute total work
    _, W_rack = rack(Q_IT_t)
    Q_coil_t, W_crah = crah(Q_IT_t)
```

```
    _, W_pumps = chilled_water(Q_coil_t, Q_IT_t)

    W_total_eff = W_rack + W_crah + W_pumps + W_comp_eff

    # Store results
    T_wb_hist.append(T_wb)
    Q_IT_hist.append(Q_IT_t)
    Q_econ_hist.append(Q_econ)
    Q_chiller_hist.append(Q_chiller_base)
    Q_chiller_eff_hist.append(Q_chiller_eff)
    Q_TES_hist.append(Q_TES)
    E_TES_hist.append(E_TES)
    W_total_base_hist.append(W_total_base)
    W_total_hist.append(W_total_eff)
    COP_base_hist.append(Q_IT_t / W_total_base)
    COP_TES_hist.append(Q_IT_t / W_total_eff)

# Convert to arrays
T_wb_hist = np.array(T_wb_hist)
Q_IT_hist = np.array(Q_IT_hist)
Q_econ_hist = np.array(Q_econ_hist)
Q_chiller_hist = np.array(Q_chiller_hist)
Q_chiller_eff_hist = np.array(Q_chiller_eff_hist)
Q_TES_hist = np.array(Q_TES_hist)
E_TES_hist = np.array(E_TES_hist)
W_total_base_hist = np.array(W_total_base_hist)
W_total_hist = np.array(W_total_hist)
COP_base_hist = np.array(COP_base_hist)
COP_TES_hist = np.array(COP_TES_hist)

SOC_hist = E_TES_hist / E_max
```

[4]:
```
# ----------------------------
# Plotting utility
# ----------------------------

BASELINE_COLOR = "blue"
TES_COLOR = "tab:orange"
SINGLE_COLOR = "tab:blue"

def add_caption(fig_num):
    plt.figtext(
        0.5, 0.01,
        f"Figure {fig_num}",
        ha='center',
        fontsize=12,
    )
```

```
    plt.subplots_adjust(bottom=0.2)

def plot_tes_results(
    t_array,
    T_wb_hist,
    Q_IT_hist,
    SOC_hist,
    Q_TES_hist,
    Q_chiller_hist,
    Q_chiller_eff_hist,
    W_total_base_hist,
    W_total_hist,
    COP_base_hist,
    COP_TES_hist,
    model_name="TES Model",
    fig_start=1
):

    plt.figure(figsize=(10, 4))
    plt.plot(t_array, T_wb_hist, color=SINGLE_COLOR)
    plt.xlabel("Time (hr)")
    plt.ylabel("T_wb (deg C)")
    plt.title(f"{model_name}: Dynamic Ambient Condition")
    plt.grid(True)
    add_caption(fig_start)
    plt.show()

    plt.figure(figsize=(10, 4))
    plt.plot(t_array, Q_IT_hist / 1e6, color=SINGLE_COLOR)
    plt.xlabel("Time (hr)")
    plt.ylabel("Q_IT (MW)")
    plt.title(f"{model_name}: Dynamic IT Thermal Load")
    plt.grid(True)
    add_caption(fig_start + 1)
    plt.show()

    plt.figure(figsize=(10, 4))
    plt.plot(t_array, SOC_hist, color=SINGLE_COLOR)
    plt.xlabel("Time (hr)")
    plt.ylabel("SOC")
    plt.title(f"{model_name}: TES State of Charge")
    plt.grid(True)
    add_caption(fig_start + 2)
    plt.show()

    plt.figure(figsize=(10, 4))
    plt.plot(t_array, Q_TES_hist / 1e6, color=SINGLE_COLOR)
```

```
    plt.axhline(0, color="black", linewidth=0.8)
    plt.xlabel("Time (hr)")
    plt.ylabel("Q_TES (MW)")
    plt.title(f"{model_name}: TES Charging / Discharging")
    plt.grid(True)
    add_caption(fig_start + 3)
    plt.show()

    plt.figure(figsize=(10, 4))
    plt.plot(t_array, Q_chiller_hist / 1e6, color=BASELINE_COLOR,␣
↪label="Baseline")
    plt.plot(t_array, Q_chiller_eff_hist / 1e6, color=TES_COLOR,␣
↪label="TES-adjusted")
    plt.xlabel("Time (hr)")
    plt.ylabel("Chiller load (MW)")
    plt.title(f"{model_name}: Chiller Load")
    plt.grid(True)
    plt.legend()
    add_caption(fig_start + 4)
    plt.show()

    plt.figure(figsize=(10, 4))
    plt.plot(t_array, W_total_base_hist / 1e6, color=BASELINE_COLOR,␣
↪label="Baseline")
    plt.plot(t_array, W_total_hist / 1e6, color=TES_COLOR, label="TES")
    plt.xlabel("Time (hr)")
    plt.ylabel("Total power (MW)")
    plt.title(f"{model_name}: Total Cooling Power")
    plt.grid(True)
    plt.legend()
    add_caption(fig_start + 5)
    plt.show()

    plt.figure(figsize=(10, 4))
    plt.plot(t_array, COP_base_hist, color=BASELINE_COLOR, label="Baseline")
    plt.plot(t_array, COP_TES_hist, color=TES_COLOR, label="TES")
    plt.xlabel("Time (hr)")
    plt.ylabel("System COP")
    plt.title(f"{model_name}: System COP")
    plt.grid(True)
    plt.legend()
    add_caption(fig_start + 6)
    plt.show()
```

[5]:
```
plot_tes_results(
    t_array,
    T_wb_hist,
```

```
    Q_IT_hist,
    SOC_hist,
    Q_TES_hist,
    Q_chiller_hist,
    Q_chiller_eff_hist,
    W_total_base_hist,
    W_total_hist,
    COP_base_hist,
    COP_TES_hist,
    model_name="TES Model 1",
    fig_start=3
)
```

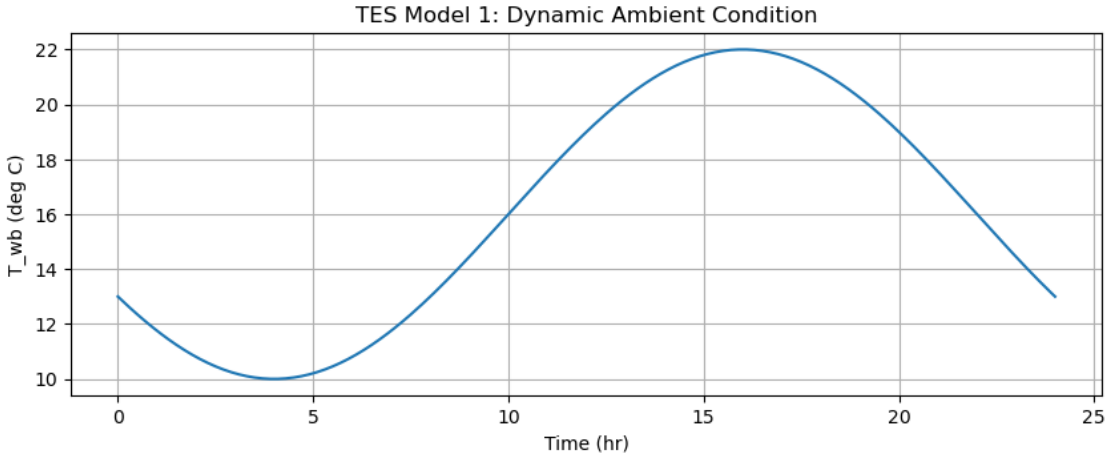

Figure 3

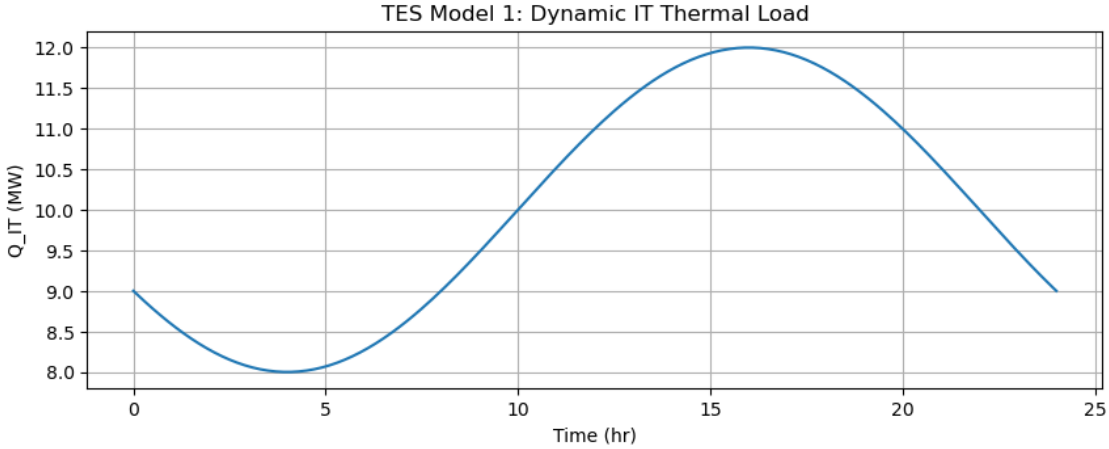

Figure 4

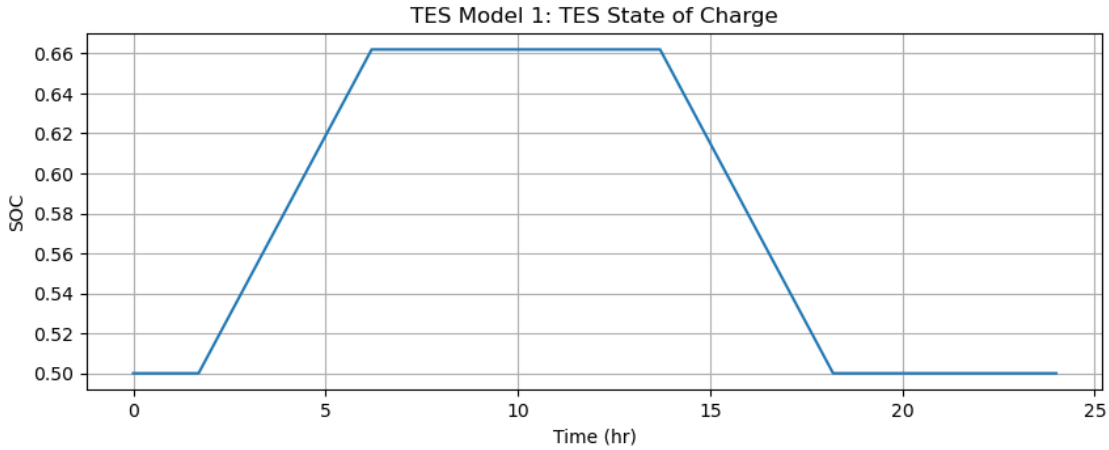

Figure 5

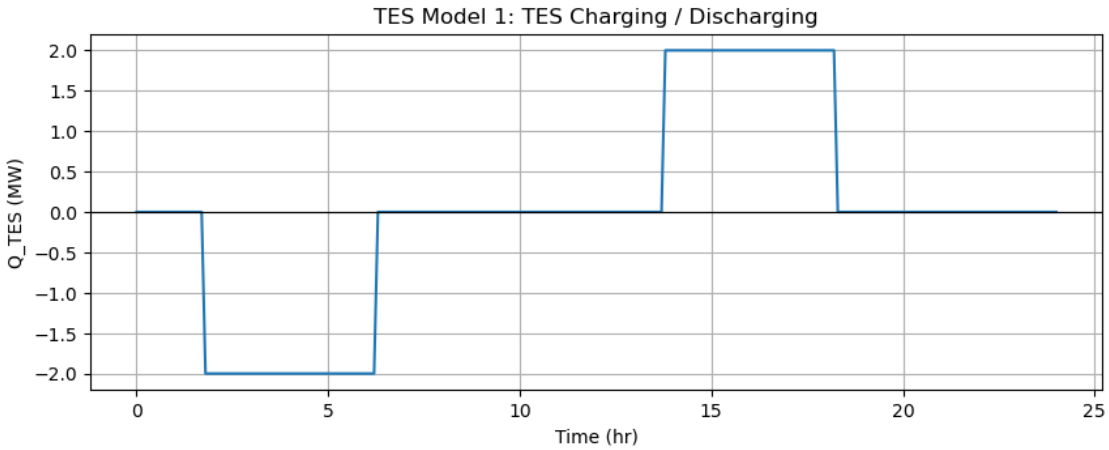

Figure 6

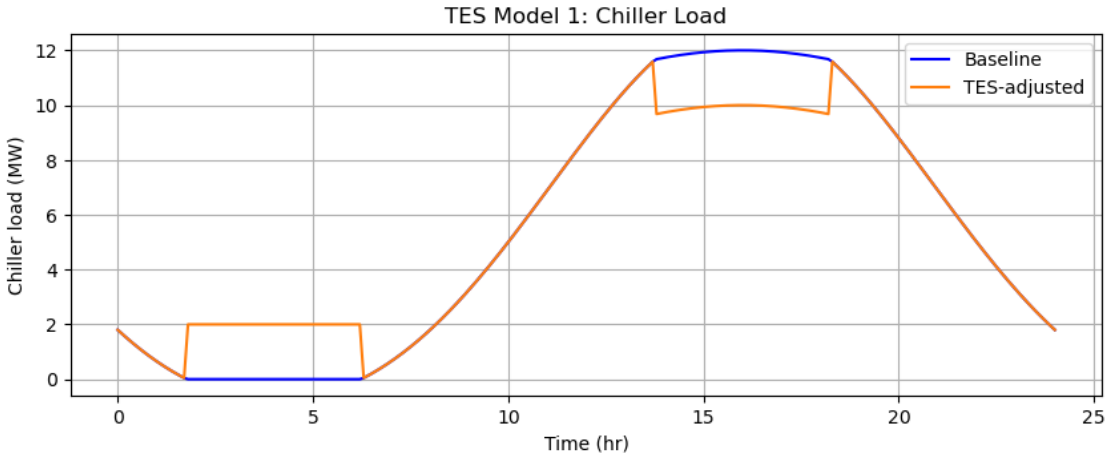

Figure 7

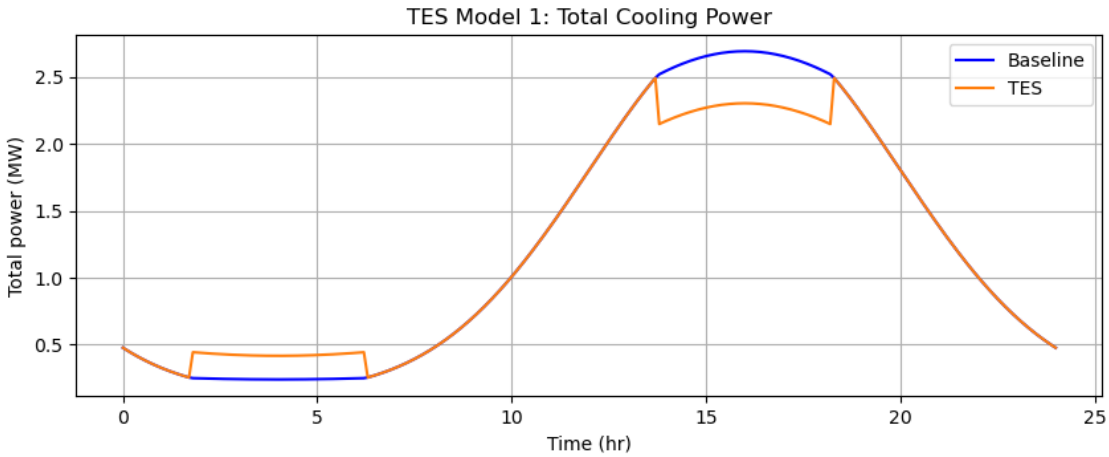

Figure 8

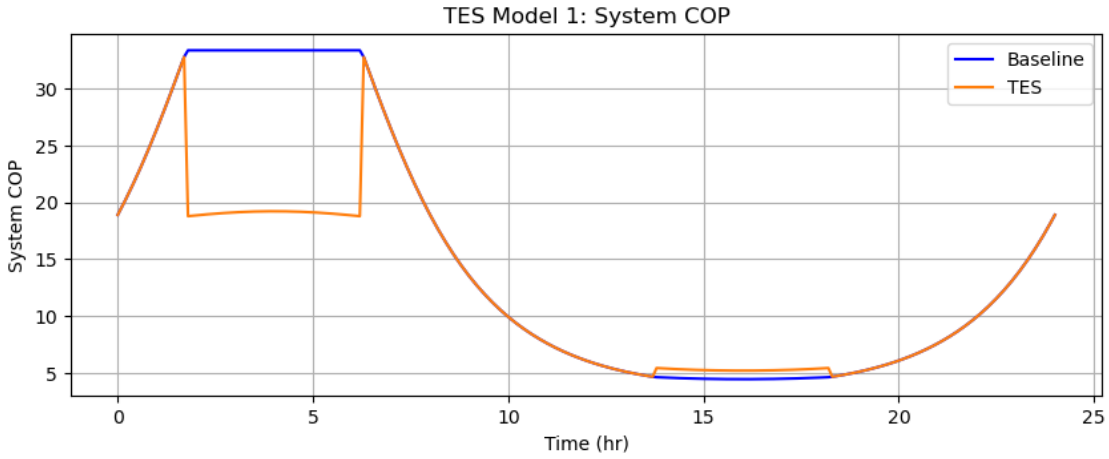

Figure 9

1.2.3 Analysis of TES Model 1

Our first model integrating TES into the data center shows clear improvements to the system. In figures 7 and 8, can see clear reductions in the peak loads achieved by charging the TES system during favorable ambient periods and discharging during high ambient temperature periods. We effectively are redistributing the cooling demand over time. This demonstrates the intended peak-shaving capability of TES.

This load shifting is also reflected in system power consumption as seen in figure 6. While TES increases power consumption during charging periods, it reduces power consumption during peak hours. Once again, keep in mind that the total energy usage throughout the day is not supposed to decrease, as we are focused on reducing peak load.

The COP shown in figure 9 further highlights this tradeoff. During peak conditions, the system with TES has a higher COP than the baseline system, indicating improved efficiency when the cooling demand and ambient temperature are highest. However, during charging periods, the COP is lower than the baseline due to the additional cooling required to store energy. This is expected as TES shifts efficiency to periods where it is most valuable.

However, this is a first-order model with limitations.

1. The control logic is very binary and idealized, as shown in figures 5 and 6. It charges whenever the wet-bulb temperature falls below the first breakpoint, discharges whenever the wet-bulb temperature exceeds the second breakpoint, and remains idle everywhere else. In practice, TES control depends on a far more complicated combination of ambient conditions, chiller load, tank state of charge, and perhaps electricity cost.
2. We ignore thermal losses associated with TES because of how we modeled the system using idealized sinusoidal functions.

1.3 Thermal Energy Storage Model 2

I will now introduce some engineering improvements.

1. One of the biggest problems with Model 1 is that it does not care about the chiller load. It discharges even when the chiller isn't contributing much. We will make TES control dependent on the chiller load. This will be done by defining a desired upper bound on the chiller load $\dot{Q}_{\text{target}}$. The baseline chiller load is

$$\dot{Q}_{\text{chiller,base}}(t) = \dot{Q}_{\text{IT}}(t) - \dot{Q}_{\text{economizer}}(t).$$

2. TES will also be dependent on ambient conditions. Below T_{charge}, cooling is efficient; above $T_{\text{discharge}}$, cooling is inefficient. We will have three regimes.

- The first is when the conditions are favorable, so we charge.
- The second is when conditions are unfavorable, so we discharge if the chiller needs help.
- The third is when condiitons are neutral. That is, there's no strong advantage to charging or discharging. Here, the baseline system will run and TES will remain idle.

$$\dot{Q}_{\text{TES,req}}(t) = \begin{cases} -\dot{Q}_{\text{TES,max}}, & T_{\text{wb}}(t) \leq T_{\text{charge}} \\ \max\left(0,\ \dot{Q}_{\text{chiller,base}}(t) - \dot{Q}_{\text{target}}\right), & T_{\text{wb}}(t) \geq T_{\text{discharge}} \\ 0, & \text{otherwise} \end{cases}$$

This requested value has constraints:

$$\dot{Q}_{\text{TES}} = \text{clip}\left(\dot{Q}_{\text{TES,req}}(t), -\dot{Q}_{\text{TES,max}}, \dot{Q}_{\text{TES,max}}\right)$$

3. We will use a simple loss model. Let λ represent the TES loss rate. Then

$$\frac{dE_{\text{TES}}}{dt} = \dot{Q}_{\text{charge}}(t) - \dot{Q}_{\text{discharge}}(t) - \lambda E_{\text{TES}}(t).$$

Or numerically,

$$E_{\text{TES}}^{t+1} = E_{\text{TES}}^{t}(1 - \lambda\Delta t) + \left(\dot{Q}_{\text{charge}}^{t} - \dot{Q}_{\text{discharge}}^{t}\right)\Delta t.$$

$$E_{\text{TES}}^{t+1} = E_{\text{TES}}^{t}(1 - \lambda\Delta t) - \dot{Q}_{\text{TES}}^{t}\Delta t.$$

Let's make sure we are still following the first law.

$$0 \leq E_{\text{TES}}^{t} \leq E_{\max}$$

For charging,

$$\left|\dot{Q}_{\text{TES}}^{t}\right| \leq \frac{E_{\max} - E_{\text{TES}}^{t}}{\Delta t}$$

For discharging,

$$\dot{Q}_{\text{TES}}^{t} \leq \frac{E_{\text{TES}}^{t}}{\Delta t}$$

We now have Model 2.

[6]:
```
# ----------------------------
# TES Model 2
# ----------------------------

# Time setup
dt = 0.1                        # hours
dt_sec = dt * 3600.0
t_array = np.arange(0, 24 + dt, dt)

# Dynamic ambient condition
T_bar = 16.0                    # deg C
A_wb = 6.0                      # deg C amplitude
phi_wb = 10.0                   # peak at 4 pm
# TES parameters
E_max = 2.0e11                  # J
E_TES = 0.5 * E_max             # start half full
Q_TES_max = 2.0e6               # W

# Improved control parameters
Q_target = 9.0e6                # W, desired chiller load
loss_rate = 0.01/24             # per hour

# Storage arrays
T_wb_hist_2 = []
Q_IT_hist_2 = []
Q_econ_hist_2 = []
Q_chiller_hist_2 = []
Q_chiller_eff_hist_2 = []
Q_TES_hist_2 = []
E_TES_hist_2 = []
W_total_base_hist_2 = []
W_total_hist_2 = []
COP_base_hist_2 = []
COP_TES_hist_2 = []
loss_energy_total = 0.0
Q_loss_hist_2 = []

for t in t_array:

    # Dynamic wet-bulb temperature
    T_wb = T_bar + A_wb * np.sin(2 * np.pi * (t - phi_wb) / 24)
```

```
    # Dynamic IT thermal load (peak at 4 PM)
    Q_IT_t = 10e6 + 2e6 * np.cos(2 * np.pi * (t - 16) / 24)

    # Run baseline system at this timestep
    Q_econ, Q_chiller_base, W_total_base = system_model(T_wb, Q_IT_t)

    # ------------------------------
    # Improved TES control
    # ------------------------------

    # Apply storage losses first
    E_loss = E_TES * loss_rate * dt
    E_TES = E_TES - E_loss
    loss_energy_total += E_loss
    Q_loss_hist_2.append(E_loss / dt_sec)

    # Hybrid TES control: efficiency-aware load targeting
    T_charge = 12.0       # deg C, charge when cooling is efficient
    T_discharge = 18.0    # deg C, discharge when cooling is inefficient

    if T_wb <= T_charge:
        # Favorable period: charge TES
        Q_TES_req = -Q_TES_max

    elif T_wb >= T_discharge:
        # Unfavorable period: discharge enough to cap chiller near Q_target
        Q_TES_req = max(0.0, Q_chiller_base - Q_target)

    else:
        # Neutral period: idle
        Q_TES_req = 0.0

    # Clip to TES hardware rate limits
    Q_TES = np.clip(Q_TES_req, -Q_TES_max, Q_TES_max)

    # Enforce energy limits
    if Q_TES > 0:  # discharging
        Q_TES = min(Q_TES, E_TES / dt_sec)
    elif Q_TES < 0:  # charging
        Q_TES = -min(abs(Q_TES), (E_max - E_TES) / dt_sec)

    # Update TES energy
    E_TES = E_TES + (-Q_TES) * dt_sec
    E_TES = max(0.0, min(E_TES, E_max))

    # Effective chiller load with TES
    Q_chiller_eff = max(Q_IT_t - Q_econ - Q_TES, 0.0)
```

```
    # Recompute chiller work using effective load
    T_cw_out, T_cond = tower_temperature(T_wb)
    W_comp_eff, Q_rej_eff = chiller(Q_chiller_eff, T_cond)

    # Recompute total work
    _, W_rack = rack(Q_IT_t)
    Q_coil_t, W_crah = crah(Q_IT_t)
    _, W_pumps = chilled_water(Q_coil_t, Q_IT_t)

    W_total_eff = W_rack + W_crah + W_pumps + W_comp_eff

    # Store results
    T_wb_hist_2.append(T_wb)
    Q_IT_hist_2.append(Q_IT_t)
    Q_econ_hist_2.append(Q_econ)
    Q_chiller_hist_2.append(Q_chiller_base)
    Q_chiller_eff_hist_2.append(Q_chiller_eff)
    Q_TES_hist_2.append(Q_TES)
    E_TES_hist_2.append(E_TES)
    W_total_base_hist_2.append(W_total_base)
    W_total_hist_2.append(W_total_eff)
    COP_base_hist_2.append(Q_IT_t / W_total_base)
    COP_TES_hist_2.append(Q_IT_t / W_total_eff)

Q_loss_hist_2 = np.array(Q_loss_hist_2)

# Convert to arrays
T_wb_hist_2 = np.array(T_wb_hist_2)
Q_IT_hist_2 = np.array(Q_IT_hist_2)
Q_econ_hist_2 = np.array(Q_econ_hist_2)
Q_chiller_hist_2 = np.array(Q_chiller_hist_2)
Q_chiller_eff_hist_2 = np.array(Q_chiller_eff_hist_2)
Q_TES_hist_2 = np.array(Q_TES_hist_2)
E_TES_hist_2 = np.array(E_TES_hist_2)
W_total_base_hist_2 = np.array(W_total_base_hist_2)
W_total_hist_2 = np.array(W_total_hist_2)
COP_base_hist_2 = np.array(COP_base_hist_2)
COP_TES_hist_2 = np.array(COP_TES_hist_2)

SOC_hist_2 = E_TES_hist_2 / E_max
```

[7]:
```
plot_tes_results(
    t_array,
    T_wb_hist_2,
    Q_IT_hist_2,
    SOC_hist_2,
```

```
    Q_TES_hist_2,
    Q_chiller_hist_2,
    Q_chiller_eff_hist_2,
    W_total_base_hist_2,
    W_total_hist_2,
    COP_base_hist_2,
    COP_TES_hist_2,
    model_name="TES Model 2",
    fig_start=10
)
```

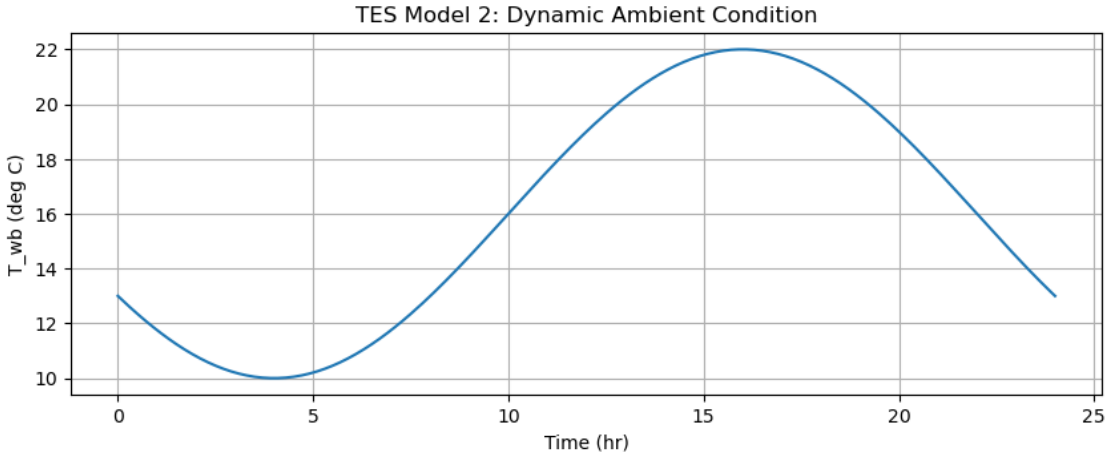

Figure 10

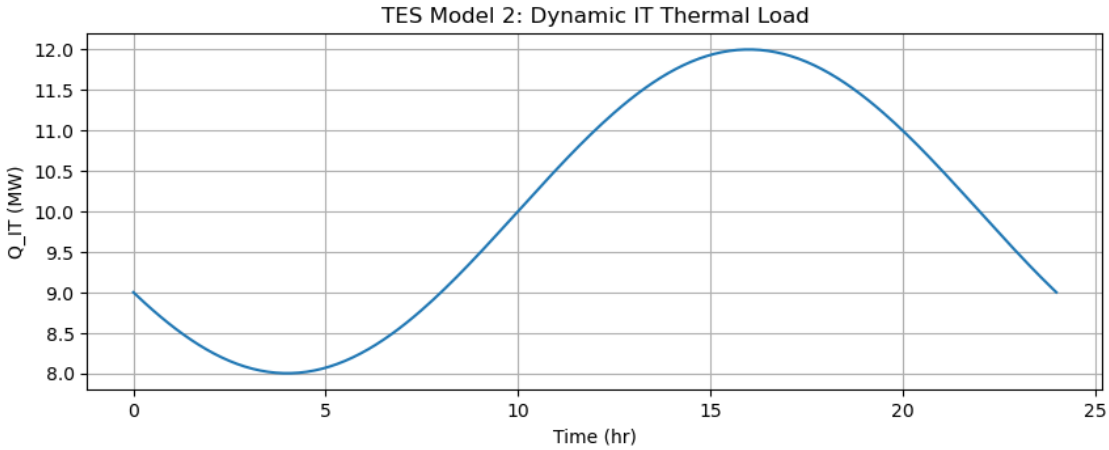

Figure 11

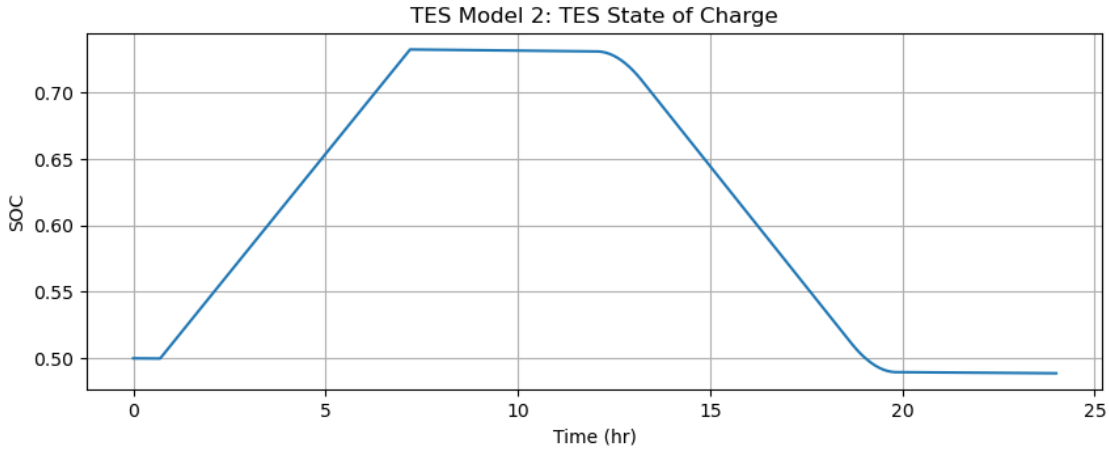

Figure 12

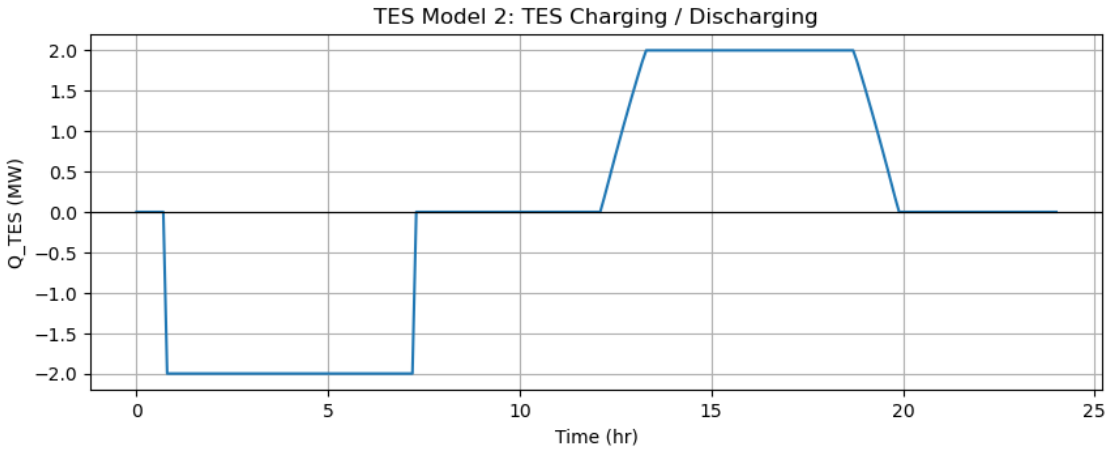

Figure 13

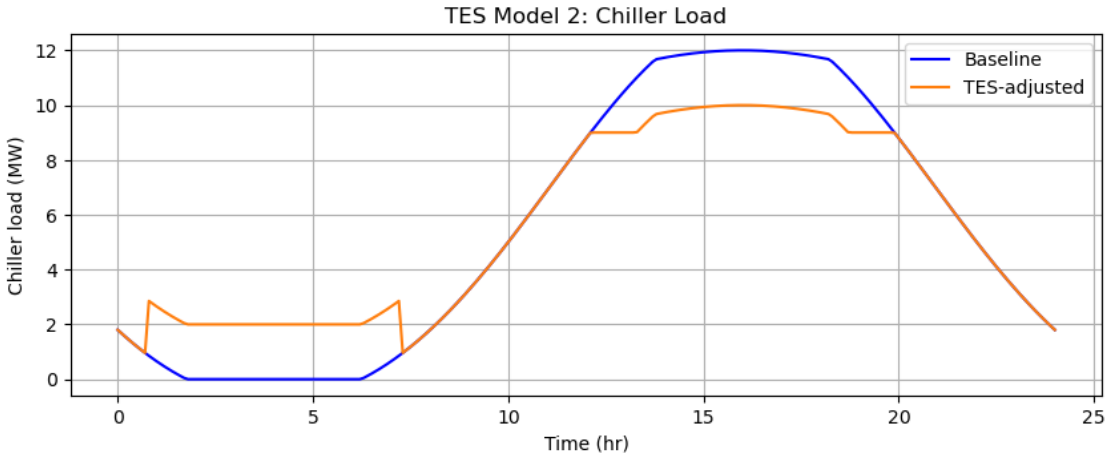

Figure 14

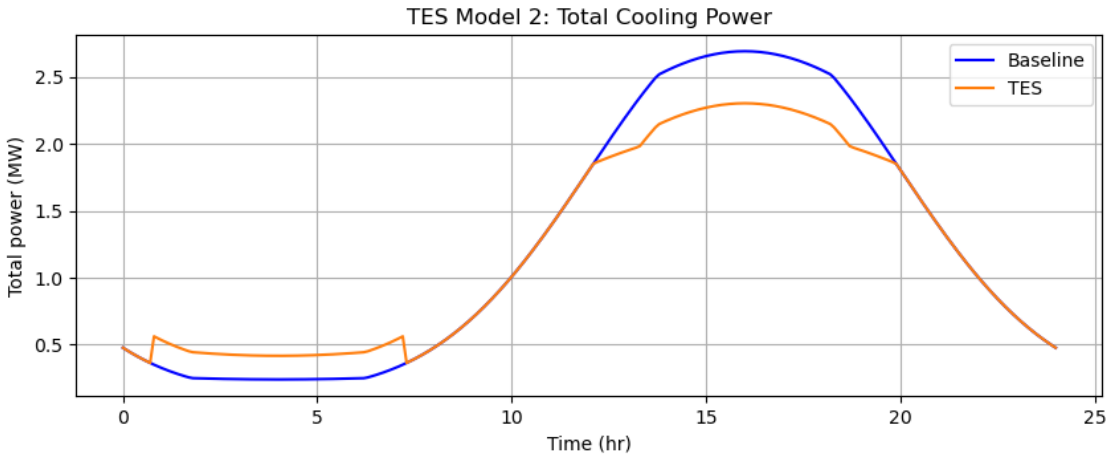

Figure 15

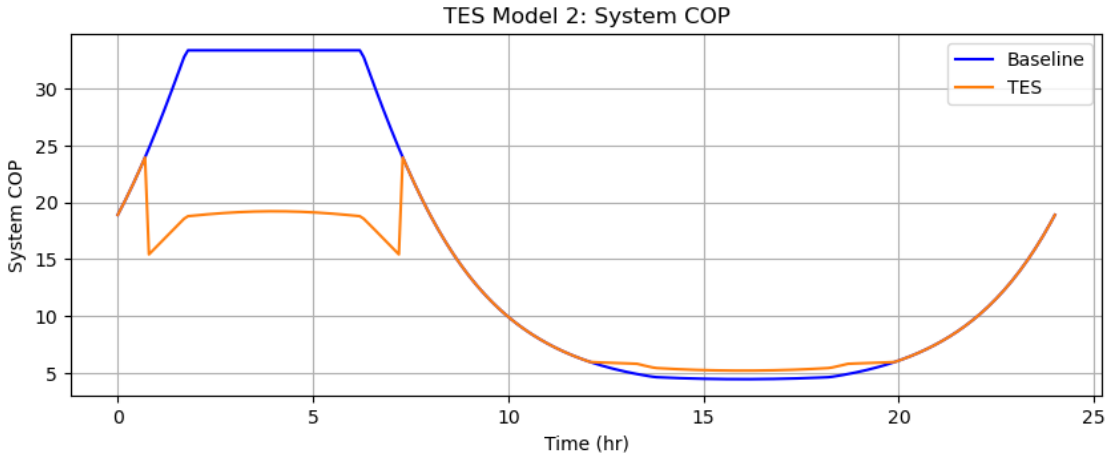

Figure 16

1.3.1 Analysis of TES Model 2

In figure 12, we can see that the thermal load has been shifted more effectively. We spend less time idle and we only discharge when we really need it.

Figure 13 shows that we start charging sooner. We then spend more time idle because there is no point in discharging too early. When the time is right, the TES begins to discharge and spends more time discharging. This is physically illustrated in figure 14. The economizer does everything at the beginning. In the early afternoon, both curves rise together, meaning the chiller is doing the work. Then, during the peak period, we see the TES kick in and thus the chiller load drops which corresponds to TES value. At night, the curves converge again.

1.4 Optimization Using a Genetic Algorithm

I predict that our model is highly dependent on a few parameters:

$$T_{\text{discharge}},\ T_{\text{charge}},\ E_{\max},\ \dot{Q}_{\text{TES,max}},\ \dot{Q}_{\text{target}}$$

To test this hypothesis, we will define a function that lets us sweep these parameters while holding the others constant, so we can see how they affect our system.

[8]:
```
def run_model_2(
    E_max,
    Q_TES_max,
    Q_target,
    T_charge,
    T_discharge,
    loss_rate=0.01/24
):
    dt = 0.1
```

```
dt_sec = dt * 3600.0
t_array = np.arange(0, 24 + dt, dt)

T_bar = 16.0
A_wb = 6.0
phi_wb = 10.0

E_TES = 0.5 * E_max

Q_chiller_eff_hist = []
W_total_hist = []
Q_IT_hist = []
SOC_hist = []

for t in t_array:
    T_wb = T_bar + A_wb * np.sin(2 * np.pi * (t - phi_wb) / 24)
    Q_IT_t = 10e6 + 2e6 * np.cos(2 * np.pi * (t - 16) / 24)

    Q_econ, Q_chiller_base, W_total_base = system_model(T_wb, Q_IT_t)

    # TES standing loss
    E_loss = E_TES * loss_rate * dt
    E_TES = E_TES - E_loss

    # Hybrid TES control
    if T_wb <= T_charge:
        Q_TES_req = -Q_TES_max

    elif T_wb >= T_discharge:
        Q_TES_req = max(0.0, Q_chiller_base - Q_target)

    else:
        Q_TES_req = 0.0

    # Power limit
    Q_TES = np.clip(Q_TES_req, -Q_TES_max, Q_TES_max)

    # Energy limits
    if Q_TES > 0:
        Q_TES = min(Q_TES, E_TES / dt_sec)
    elif Q_TES < 0:
        Q_TES = -min(abs(Q_TES), (E_max - E_TES) / dt_sec)

    # Update TES energy
    E_TES = E_TES + (-Q_TES) * dt_sec
    E_TES = max(0.0, min(E_TES, E_max))
```

```
        # Effective chiller load
        Q_chiller_eff = max(Q_IT_t - Q_econ - Q_TES, 0.0)

        # Recompute power
        T_cw_out, T_cond = tower_temperature(T_wb)
        W_comp_eff, Q_rej_eff = chiller(Q_chiller_eff, T_cond)

        _, W_rack = rack(Q_IT_t)
        Q_coil_t, W_crah = crah(Q_IT_t)
        _, W_pumps = chilled_water(Q_coil_t, Q_IT_t)

        W_total_eff = W_rack + W_crah + W_pumps + W_comp_eff

        Q_chiller_eff_hist.append(Q_chiller_eff)
        W_total_hist.append(W_total_eff)
        Q_IT_hist.append(Q_IT_t)
        SOC_hist.append(E_TES / E_max)

    Q_chiller_eff_hist = np.array(Q_chiller_eff_hist)
    W_total_hist = np.array(W_total_hist)
    Q_IT_hist = np.array(Q_IT_hist)
    SOC_hist = np.array(SOC_hist)

    peak_chiller = np.max(Q_chiller_eff_hist)
    peak_power = np.max(W_total_hist)
    overall_COP = np.sum(Q_IT_hist) / np.sum(W_total_hist)
    min_SOC = np.min(SOC_hist)
    final_SOC = SOC_hist[-1]

    return peak_chiller, peak_power, overall_COP, min_SOC, final_SOC
```

[9]:
```
# ----------------------------
# Sweep T_discharge
# ----------------------------

T_discharge_vals = np.linspace(16, 22, 8)

peak_chiller_vals = []
peak_power_vals = []
overall_COP_vals = []
min_SOC_vals = []

for T_discharge in T_discharge_vals:
    peak_chiller, peak_power, overall_COP, min_SOC, final_SOC = run_model_2(
        E_max=2.0e11,
        Q_TES_max=2.0e6,
        Q_target=9.0e6,
```

```
        T_charge=12.0,
        T_discharge=T_discharge
    )

    peak_chiller_vals.append(peak_chiller / 1e6)
    peak_power_vals.append(peak_power / 1e6)
    overall_COP_vals.append(overall_COP)
    min_SOC_vals.append(min_SOC)

fig_counter = 17

# Peak chiller
plt.figure(figsize=(10,4))
plt.plot(T_discharge_vals, peak_chiller_vals, marker='o')
plt.xlabel("T_discharge (°C)")
plt.ylabel("Peak chiller load (MW)")
plt.title("Peak Chiller Load vs T_discharge")
plt.grid(True)
add_caption(fig_counter); fig_counter += 1
plt.show()

# Peak power
plt.figure(figsize=(10,4))
plt.plot(T_discharge_vals, peak_power_vals, marker='o')
plt.xlabel("T_discharge (°C)")
plt.ylabel("Peak power (MW)")
plt.title("Peak Power vs T_discharge")
plt.grid(True)
add_caption(fig_counter); fig_counter += 1
plt.show()

# COP
plt.figure(figsize=(10,4))
plt.plot(T_discharge_vals, overall_COP_vals, marker='o')
plt.xlabel("T_discharge (°C)")
plt.ylabel("Overall COP")
plt.title("COP vs T_discharge")
plt.grid(True)
add_caption(fig_counter); fig_counter += 1
plt.show()

# Min SOC
plt.figure(figsize=(10,4))
plt.plot(T_discharge_vals, min_SOC_vals, marker='o')
plt.xlabel("T_discharge (°C)")
plt.ylabel("Minimum SOC")
plt.title("Minimum SOC vs T_discharge")
```

```
plt.grid(True)
add_caption(fig_counter); fig_counter += 1
plt.show()
```

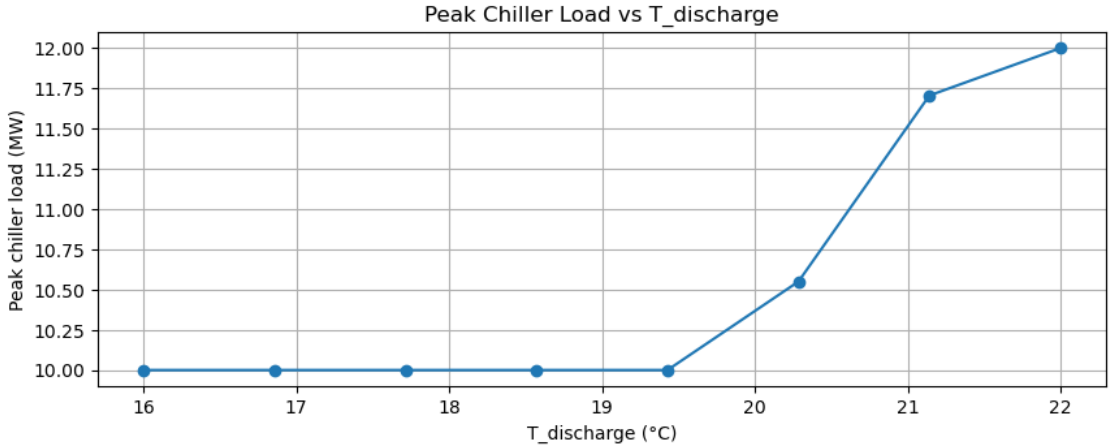

Figure 17

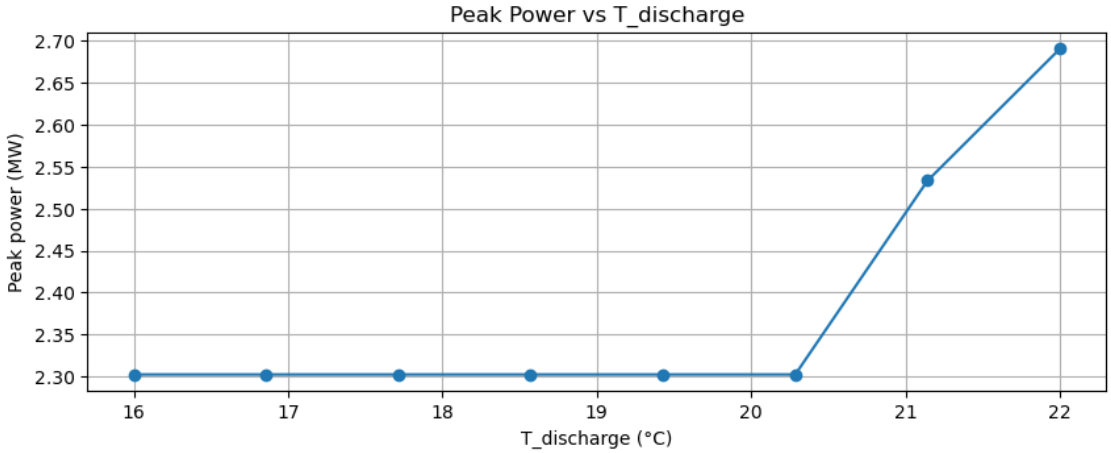

Figure 18

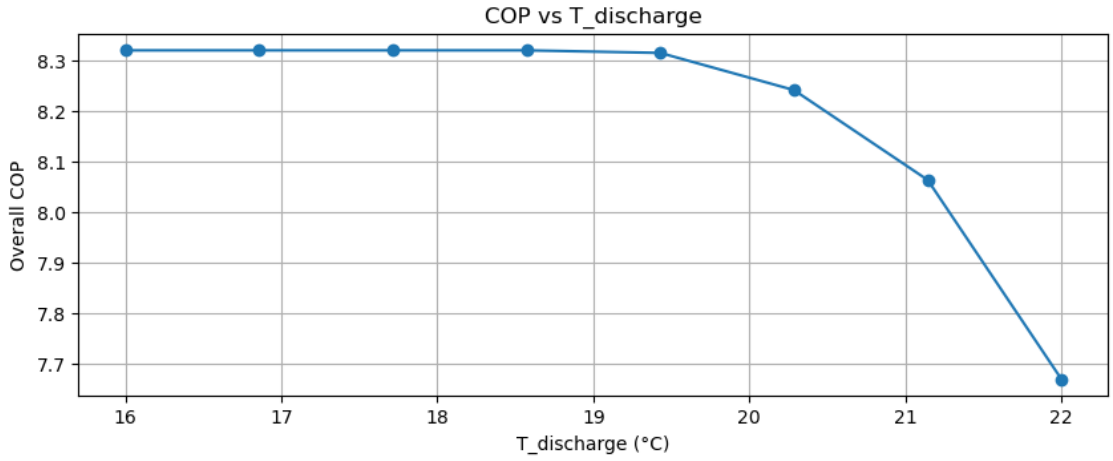

Figure 19

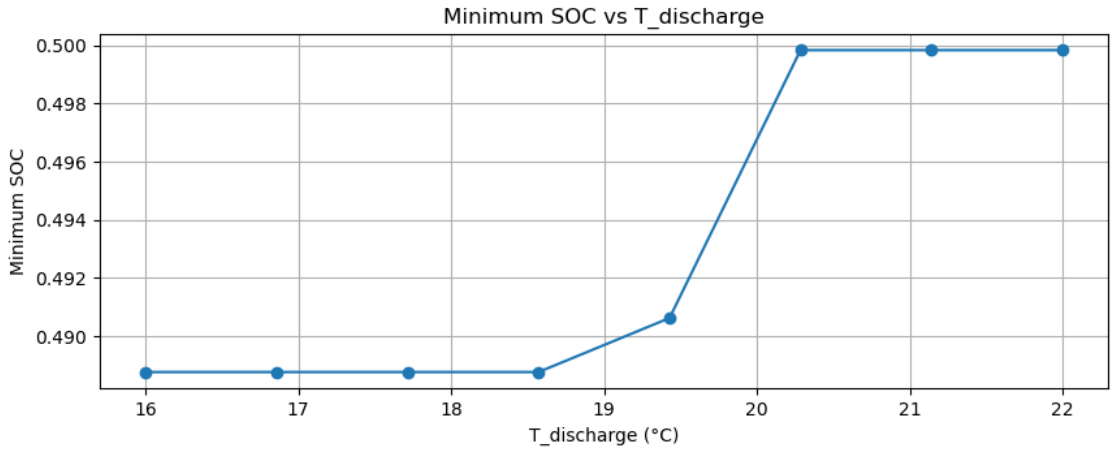

Figure 20

As long as we keep $T_{\text{discharge}}$ below 20°C, performance remains fine.

[10]:
```
# ----------------------------
# Sweep T_charge
# ----------------------------

T_charge_vals = np.linspace(8, 14, 8)

peak_chiller_vals = []
peak_power_vals = []
overall_COP_vals = []
min_SOC_vals = []
```

```
for T_charge in T_charge_vals:
    peak_chiller, peak_power, overall_COP, min_SOC, final_SOC = run_model_2(
        E_max=2.0e11,
        Q_TES_max=2.0e6,
        Q_target=9.0e6,
        T_charge=T_charge,
        T_discharge=18.0     # use best from previous sweep
    )

    peak_chiller_vals.append(peak_chiller / 1e6)
    peak_power_vals.append(peak_power / 1e6)
    overall_COP_vals.append(overall_COP)
    min_SOC_vals.append(min_SOC)

# Peak chiller
plt.figure(figsize=(10,4))
plt.plot(T_charge_vals, peak_chiller_vals, marker='o')
plt.xlabel("T_charge (°C)")
plt.ylabel("Peak chiller load (MW)")
plt.title("Peak Chiller Load vs T_charge")
plt.grid(True)
add_caption(fig_counter); fig_counter += 1
plt.show()

# Peak power
plt.figure(figsize=(10,4))
plt.plot(T_charge_vals, peak_power_vals, marker='o')
plt.xlabel("T_charge (°C)")
plt.ylabel("Peak power (MW)")
plt.title("Peak Power vs T_charge")
plt.grid(True)
add_caption(fig_counter); fig_counter += 1
plt.show()

# COP
plt.figure(figsize=(10,4))
plt.plot(T_charge_vals, overall_COP_vals, marker='o')
plt.xlabel("T_charge (°C)")
plt.ylabel("Overall COP")
plt.title("COP vs T_charge")
plt.grid(True)
add_caption(fig_counter); fig_counter += 1
plt.show()

# Min SOC
plt.figure(figsize=(10,4))
plt.plot(T_charge_vals, min_SOC_vals, marker='o')
```

```
plt.xlabel("T_charge (°C)")
plt.ylabel("Minimum SOC")
plt.title("Minimum SOC vs T_charge")
plt.grid(True)
add_caption(fig_counter); fig_counter += 1
plt.show()
```

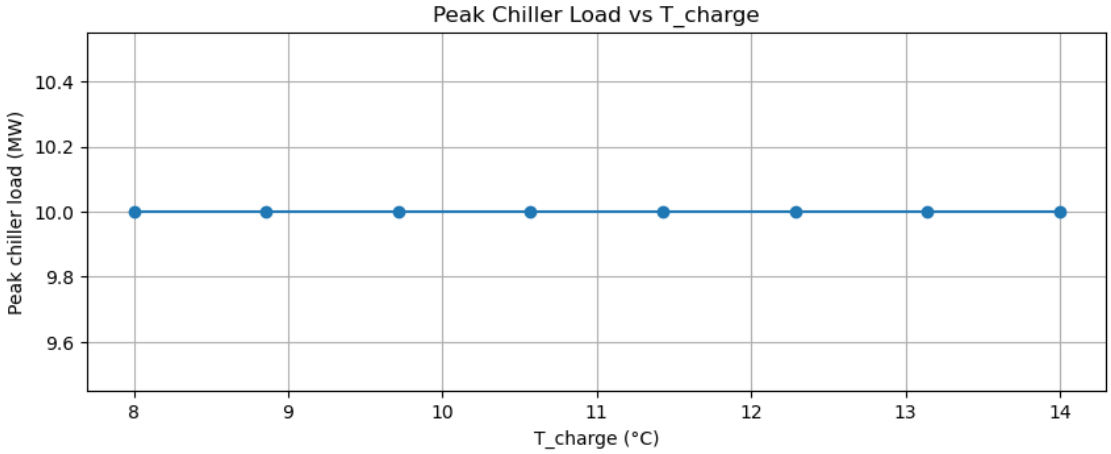

Figure 21

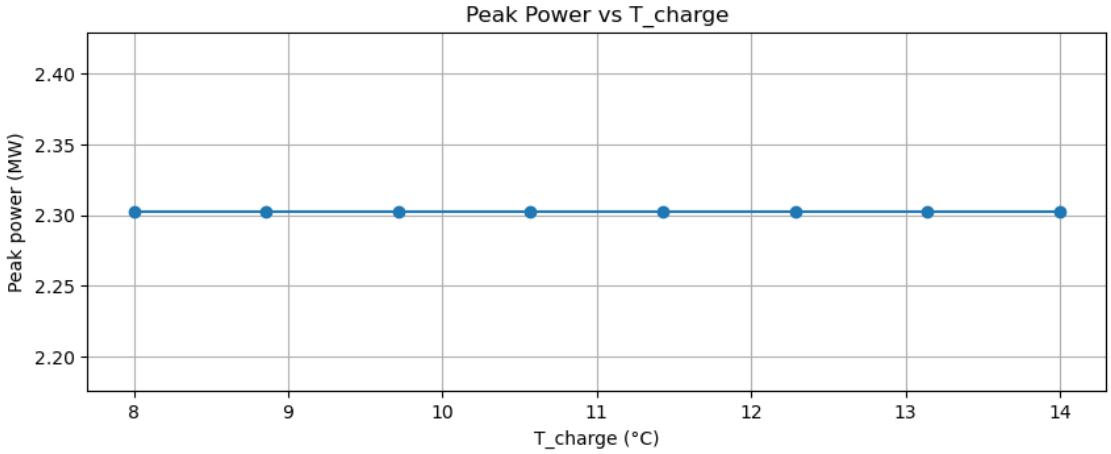

Figure 22

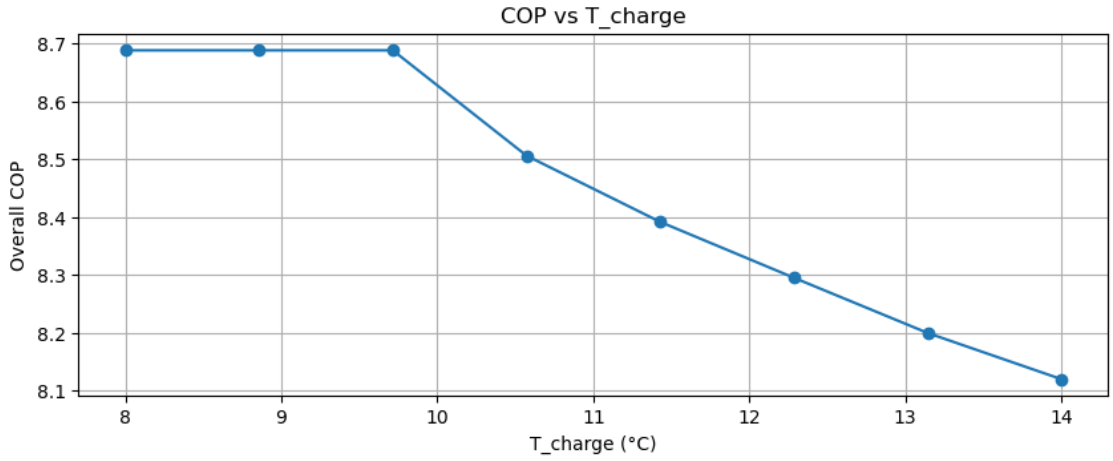

Figure 23

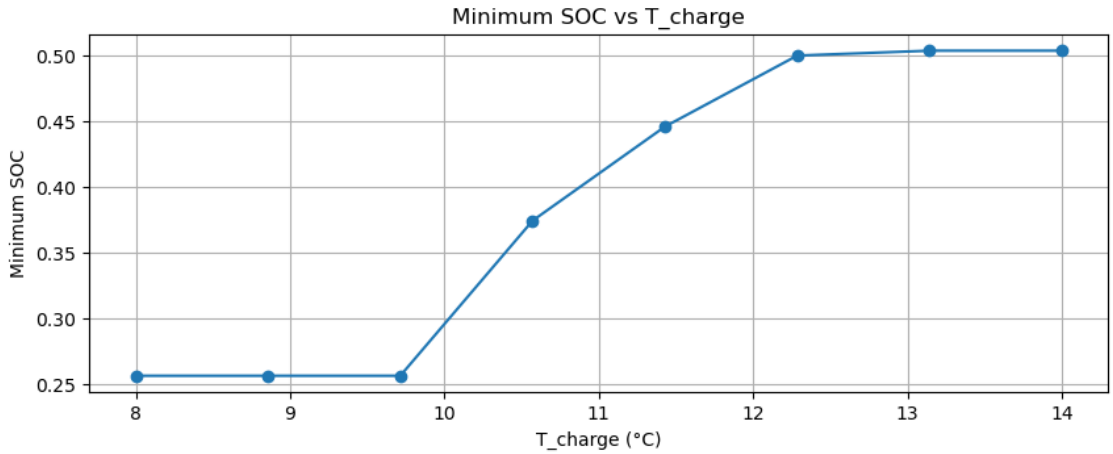

Figure 24

T_{charge} has no impact on peak chiller or power. However, as we can see in figure 41, it does affect COP. The sooner we charge, the more value we get from the TES. In figure 42, we can see that minimum TES is quite low at low values of T_{charge}. At higher values of T_{charge}, there is potentially too much underutilized charge.

[11]:
```
# ----------------------------
# Sweep E_max
# ----------------------------

E_max_vals = np.linspace(0.5e11, 5e11, 10)  # Joules

peak_chiller_vals = []
peak_power_vals = []
```

```
overall_COP_vals = []
min_SOC_vals = []
final_SOC_vals = []

for E_max in E_max_vals:

    peak_chiller, peak_power, overall_COP, min_SOC, final_SOC = run_model_2(
        E_max=E_max,
        Q_TES_max=2.0e6,
        Q_target=9.0e6,
        T_charge=12.0,
        T_discharge=18.0
    )

    peak_chiller_vals.append(peak_chiller / 1e6)
    peak_power_vals.append(peak_power / 1e6)
    overall_COP_vals.append(overall_COP)
    min_SOC_vals.append(min_SOC)
    final_SOC_vals.append(final_SOC)

# Peak chiller
plt.figure()
plt.plot(E_max_vals / 1e11, peak_chiller_vals, marker='o')
plt.xlabel("E_max (×10^11 J)")
plt.ylabel("Peak chiller load (MW)")
plt.title("Effect of TES Capacity on Peak Chiller Load")
plt.grid(True)
add_caption(fig_counter); fig_counter += 1
plt.show()

# Peak power
plt.figure()
plt.plot(E_max_vals / 1e11, peak_power_vals, marker='o')
plt.xlabel("E_max (×10^11 J)")
plt.ylabel("Peak total power (MW)")
plt.title("Effect of TES Capacity on Peak Power")
plt.grid(True)
add_caption(fig_counter); fig_counter += 1
plt.show()

# COP
plt.figure()
plt.plot(E_max_vals / 1e11, overall_COP_vals, marker='o')
plt.xlabel("E_max (×10^11 J)")
plt.ylabel("Overall COP")
plt.title("Effect of TES Capacity on System COP")
plt.grid(True)
```

```
add_caption(fig_counter); fig_counter += 1
plt.show()

# SOC check
plt.figure()
plt.plot(E_max_vals / 1e11, min_SOC_vals, marker='o', label="Min SOC")
plt.plot(E_max_vals / 1e11, final_SOC_vals, marker='o', label="Final SOC")
plt.xlabel("E_max (×10^11 J)")
plt.ylabel("SOC")
plt.title("SOC Behavior vs TES Capacity")
plt.grid(True)
plt.legend()
add_caption(fig_counter); fig_counter += 1
plt.show()
```

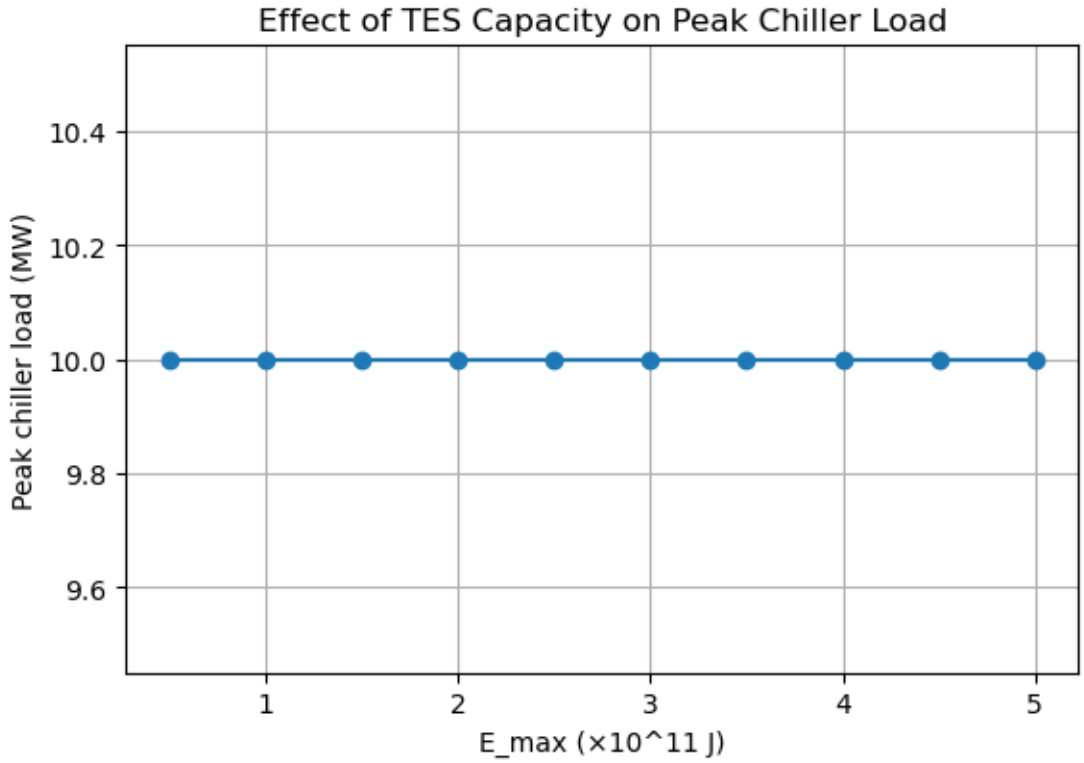

Figure 25

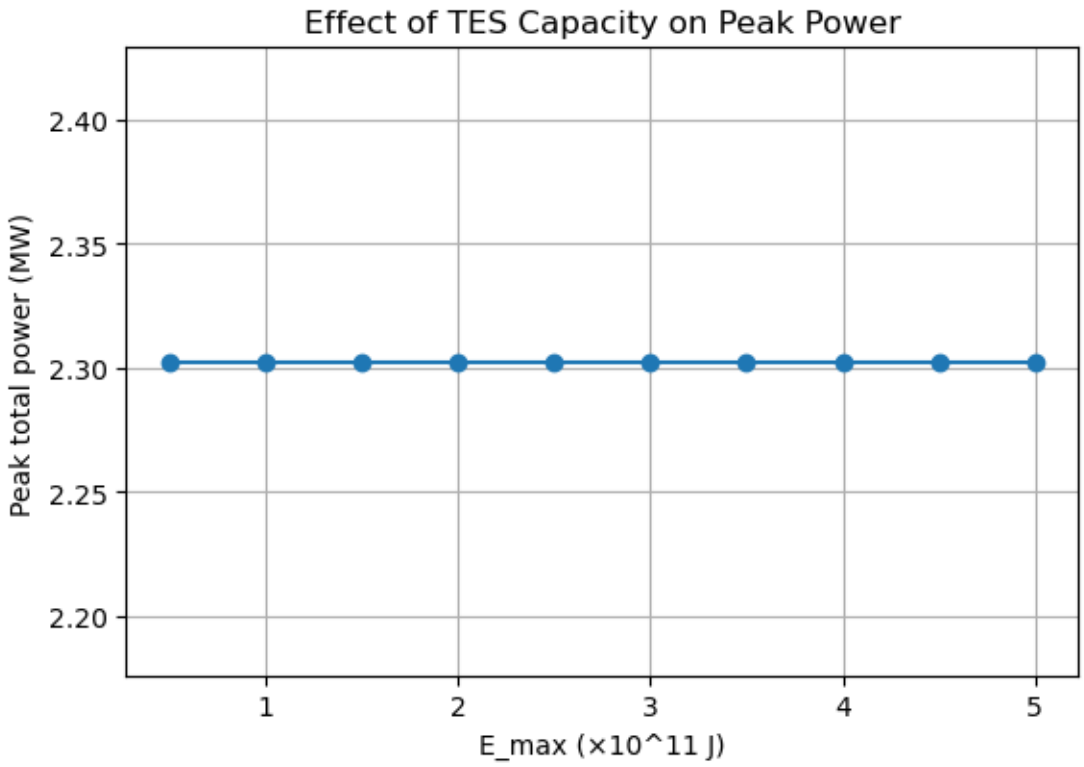
Effect of TES Capacity on Peak Power
Peak total power (MW)
2.40
2.35
2.30
2.25
2.20
1
2
3
4
5
E_max (×10^11 J)

Figure 26

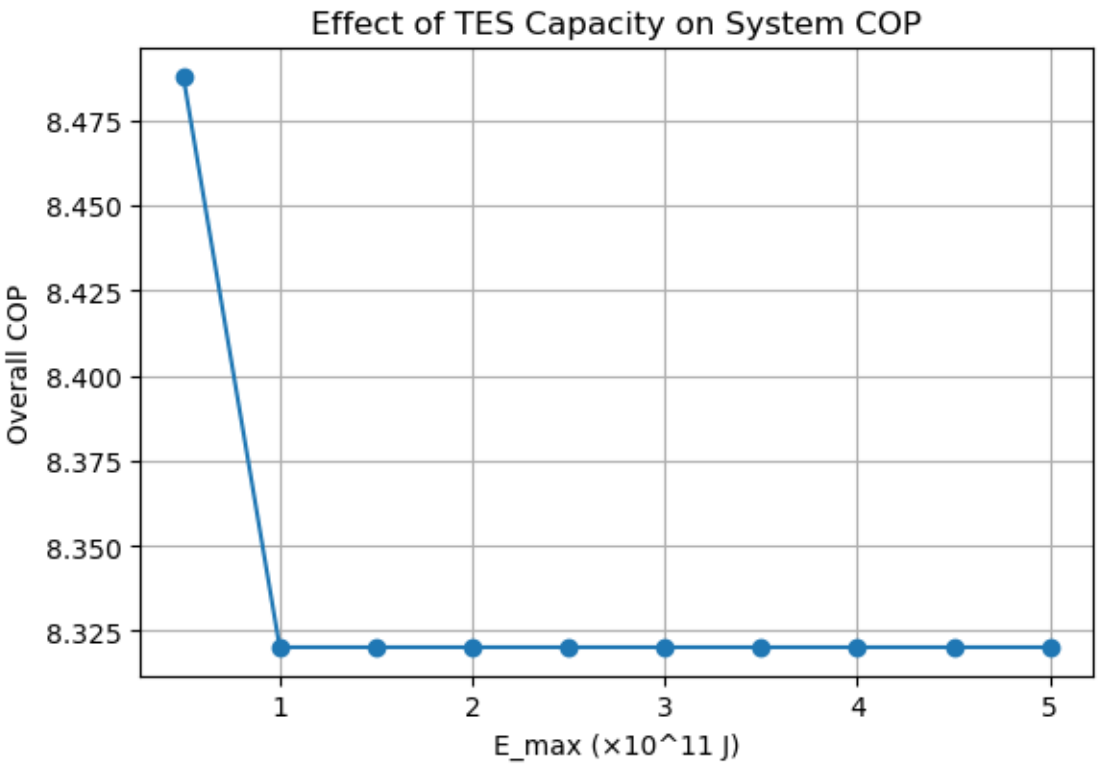
Effect of TES Capacity on System COP
Overall COP
8.475
8.450
8.425
8.400
8.375
8.350
8.325
1
2
3
4
5
E_max (×10^11 J)

Figure 27

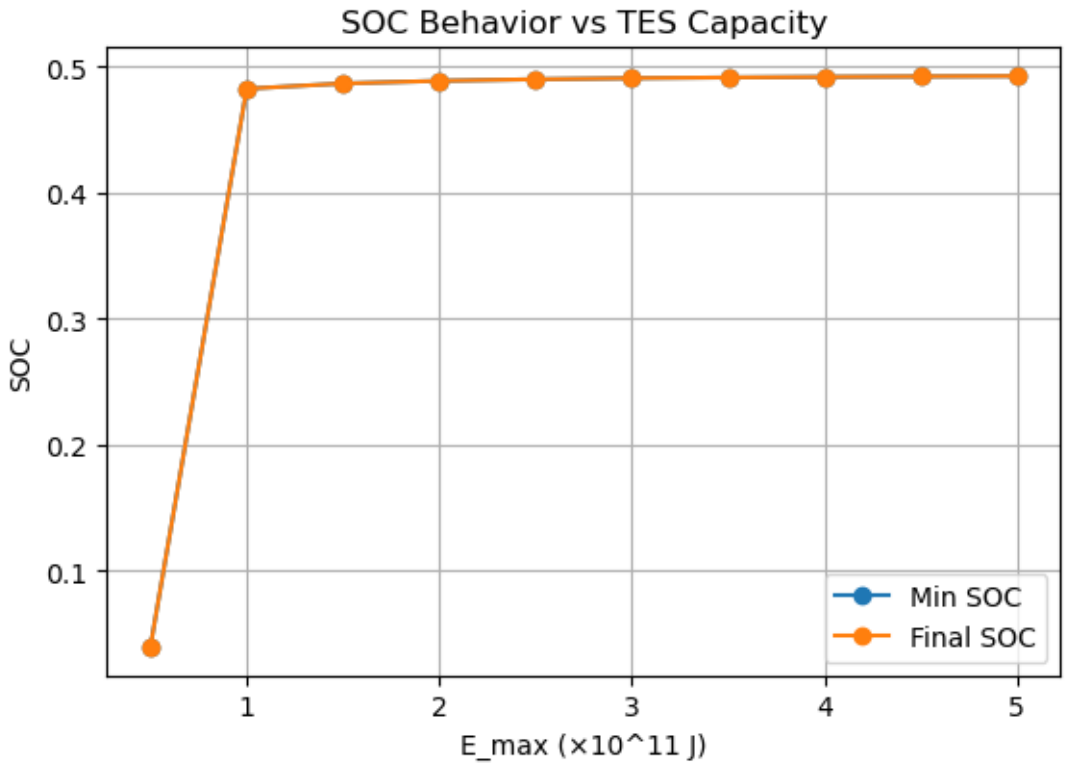

Figure 28

$E_{\max}$ has no effect on peak chiller load or peak total power. This tells us that the system is not limited by energy capacity because once there is sufficient power to support the discharge, the total no longer matters, and there are other things limiting us. Figure 31 also tells us that the change in COP is very small. We will stick with the value we currently have.

[12]:
```
# ----------------------------
# Sweep Q_TES_max (TES power capacity)
# ----------------------------

Q_TES_max_vals = np.linspace(0.5e6, 4.0e6, 10)  # 0.5-4 MW

peak_chiller_vals = []
peak_power_vals = []
overall_COP_vals = []
min_SOC_vals = []
final_SOC_vals = []

for Q_TES_max in Q_TES_max_vals:

    peak_chiller, peak_power, overall_COP, min_SOC, final_SOC = run_model_2(
```

```
        E_max=2.0e11,
        Q_TES_max=Q_TES_max,
        Q_target=9.0e6,
        T_charge=12.0,
        T_discharge=18.0
    )

    peak_chiller_vals.append(peak_chiller / 1e6)
    peak_power_vals.append(peak_power / 1e6)
    overall_COP_vals.append(overall_COP)
    min_SOC_vals.append(min_SOC)
    final_SOC_vals.append(final_SOC)

# Peak chiller load
plt.figure()
plt.plot(Q_TES_max_vals / 1e6, peak_chiller_vals, marker='o')
plt.xlabel("Q_TES_max (MW)")
plt.ylabel("Peak chiller load (MW)")
plt.title("Effect of TES Power Capacity on Peak Chiller Load")
plt.grid(True)
add_caption(fig_counter); fig_counter += 1
plt.show()

# Peak total power
plt.figure()
plt.plot(Q_TES_max_vals / 1e6, peak_power_vals, marker='o')
plt.xlabel("Q_TES_max (MW)")
plt.ylabel("Peak total power (MW)")
plt.title("Effect of TES Power Capacity on Peak Total Power")
plt.grid(True)
add_caption(fig_counter); fig_counter += 1
plt.show()

# Overall COP
plt.figure()
plt.plot(Q_TES_max_vals / 1e6, overall_COP_vals, marker='o')
plt.xlabel("Q_TES_max (MW)")
plt.ylabel("Overall COP")
plt.title("Effect of TES Power Capacity on System COP")
plt.grid(True)
add_caption(fig_counter); fig_counter += 1
plt.show()

# SOC behavior
plt.figure()
plt.plot(Q_TES_max_vals / 1e6, min_SOC_vals, marker='o', label="Min SOC")
plt.plot(Q_TES_max_vals / 1e6, final_SOC_vals, marker='o', label="Final SOC")
```

```
plt.xlabel("Q_TES_max (MW)")
plt.ylabel("SOC")
plt.title("SOC Behavior vs TES Power Capacity")
plt.grid(True)
plt.legend()
add_caption(fig_counter); fig_counter += 1
plt.show()
```

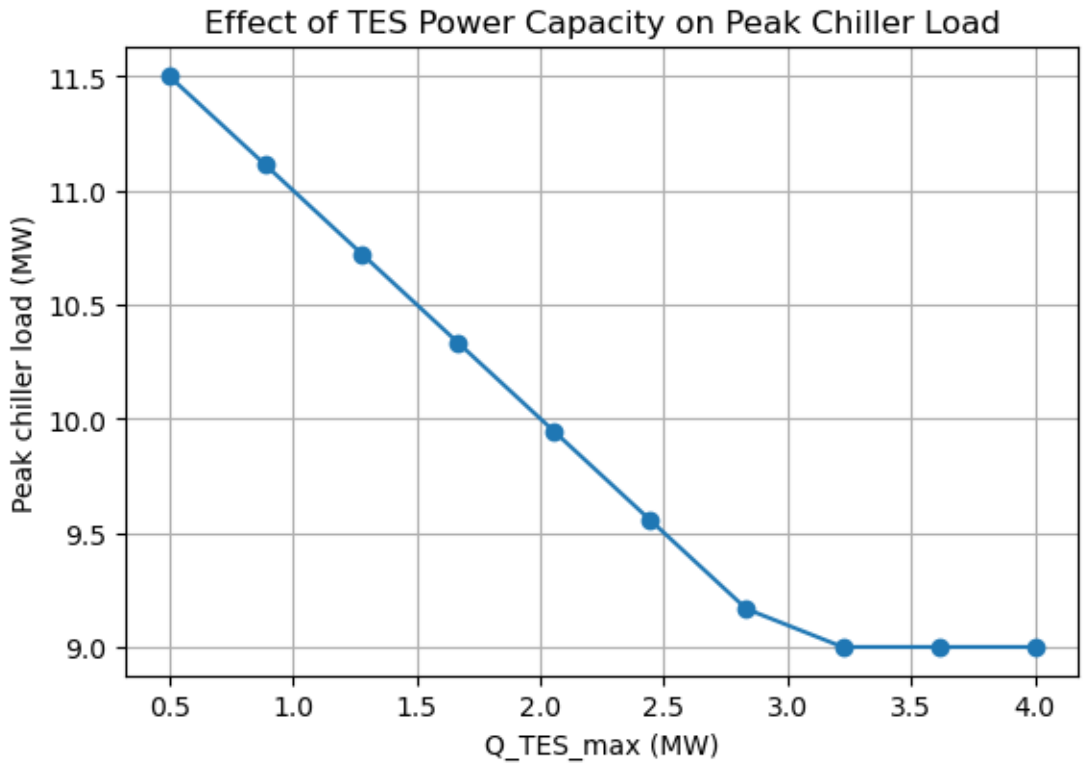

Figure 29

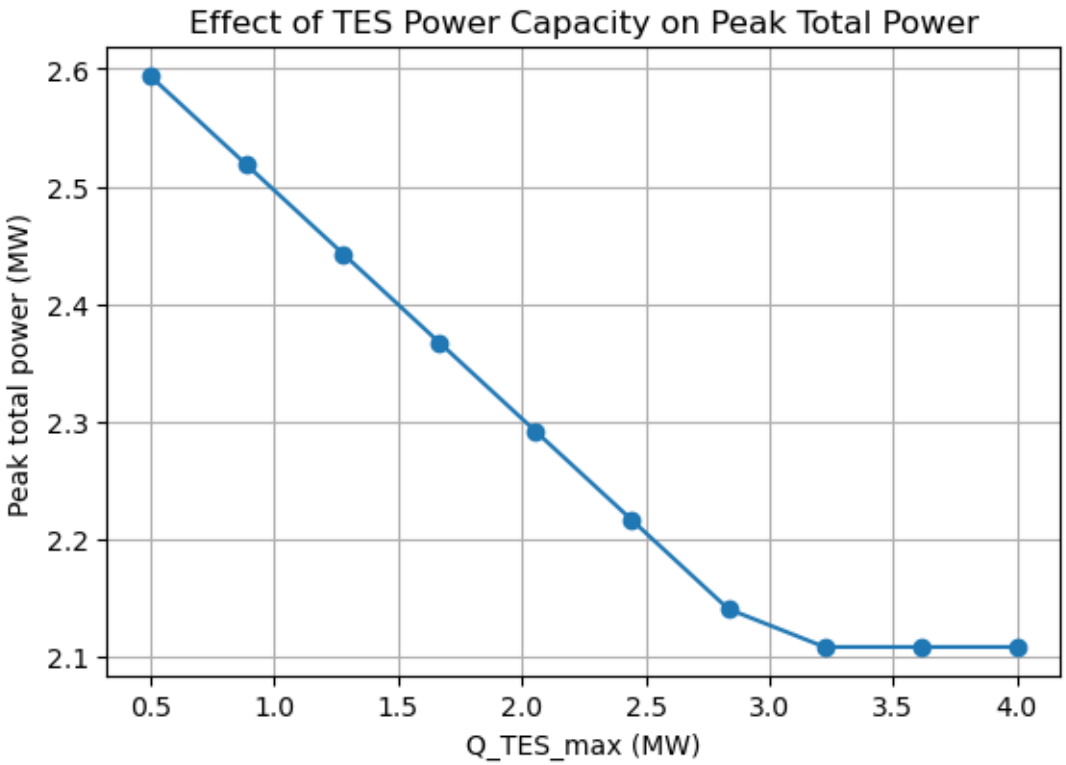
Effect of TES Power Capacity on Peak Total Power
Peak total power (MW)
2.6
2.5
2.4
2.3
2.2
2.1
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Q_TES_max (MW)

Figure 30

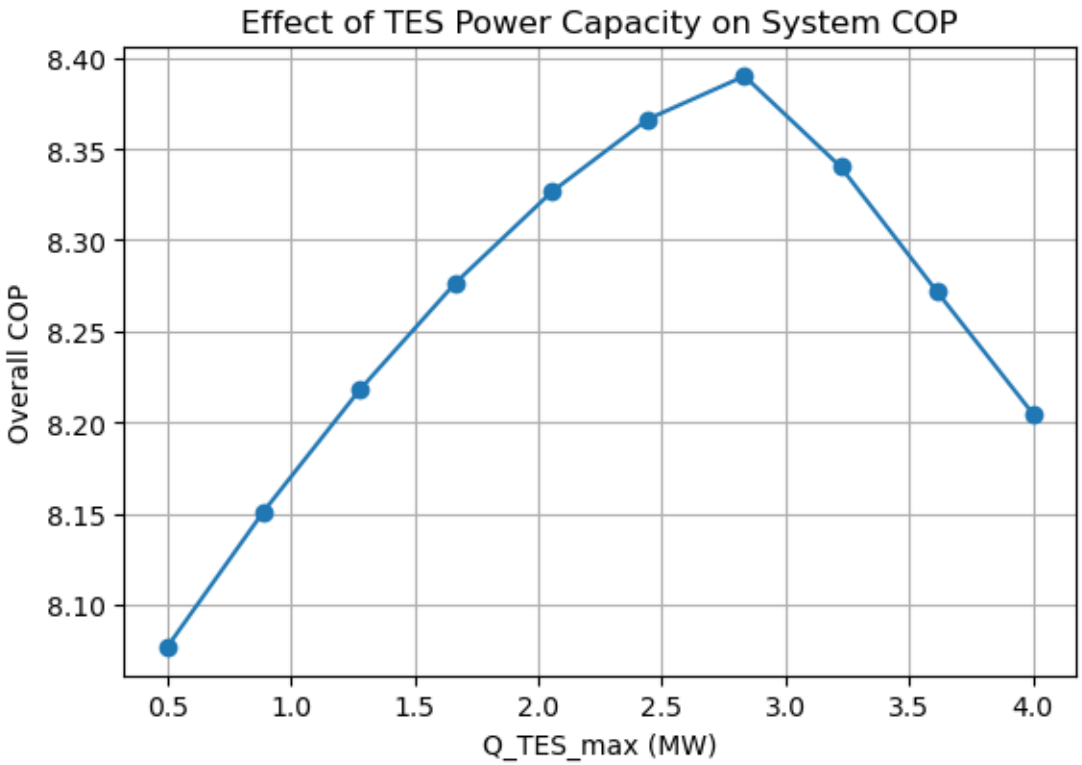
Effect of TES Power Capacity on System COP
Overall COP
8.40
8.35
8.30
8.25
8.20
8.15
8.10
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Q_TES_max (MW)

Figure 31

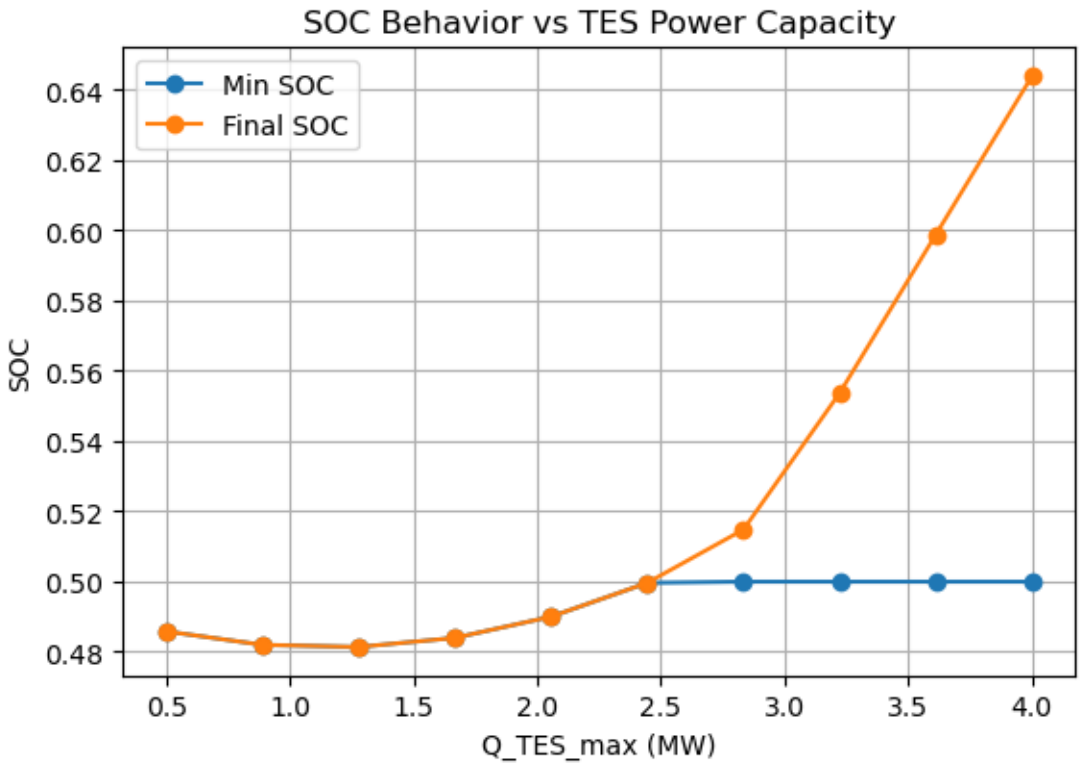

Figure 32

Our third parameter swept, $\dot{Q}_{\text{TES,max}}$ is very revealing. As seen in figure 31, increasing it also increases the peak-shaving ability up to a certain point. Figure 32 shows the same trend but from a total power perspective.

In figure 33, we see COP increasing up to a certain point and then decreasing once the TES gets too aggressive. We aren't really building our system around COP, but this is valuable insight.

Finally, figure 34 tells us that there are no values that will create bottlenecks in charging.

[13]:
```
# ----------------------------
# Sweep Q_target
# ----------------------------

Q_target_vals = np.linspace(6e6, 11e6, 11)

peak_chiller_vals = []
peak_power_vals = []
overall_COP_vals = []
min_SOC_vals = []
final_SOC_vals = []
```

```
for Q_target in Q_target_vals:
    peak_chiller, peak_power, overall_COP, min_SOC, final_SOC = run_model_2(
        E_max=2.0e11,
        Q_TES_max=2.8e6,
        Q_target=Q_target,
        T_charge=11.0,
        T_discharge=18.0
    )

    peak_chiller_vals.append(peak_chiller / 1e6)
    peak_power_vals.append(peak_power / 1e6)
    overall_COP_vals.append(overall_COP)
    min_SOC_vals.append(min_SOC)
    final_SOC_vals.append(final_SOC)

plt.figure(figsize=(10,4))
plt.plot(Q_target_vals / 1e6, peak_chiller_vals, marker='o')
plt.xlabel("Q_target (MW)")
plt.ylabel("Peak chiller load (MW)")
plt.title("Sensitivity of Peak Chiller Load to Q_target")
plt.grid(True)
add_caption(fig_counter)
fig_counter += 1
plt.show()

plt.figure(figsize=(10,4))
plt.plot(Q_target_vals / 1e6, peak_power_vals, marker='o')
plt.xlabel("Q_target (MW)")
plt.ylabel("Peak total power (MW)")
plt.title("Sensitivity of Peak Total Power to Q_target")
plt.grid(True)
add_caption(fig_counter)
fig_counter += 1
plt.show()

plt.figure(figsize=(10,4))
plt.plot(Q_target_vals / 1e6, overall_COP_vals, marker='o')
plt.xlabel("Q_target (MW)")
plt.ylabel("Overall COP")
plt.title("Sensitivity of Overall COP to Q_target")
plt.grid(True)
add_caption(fig_counter)
fig_counter += 1
plt.show()

plt.figure(figsize=(10,4))
plt.plot(Q_target_vals / 1e6, min_SOC_vals, marker='o', label="Minimum SOC")
```

```
plt.plot(Q_target_vals / 1e6, final_SOC_vals, marker='o', label="Final SOC")
plt.xlabel("Q_target (MW)")
plt.ylabel("SOC")
plt.title("Sensitivity of SOC to Q_target")
plt.grid(True)
plt.legend()
add_caption(fig_counter)
fig_counter += 1
plt.show()
```

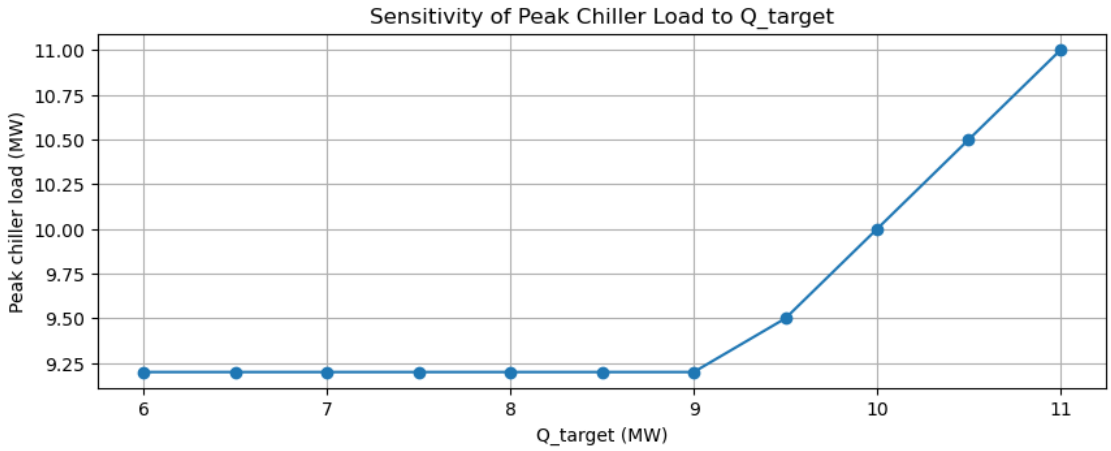

Figure 33

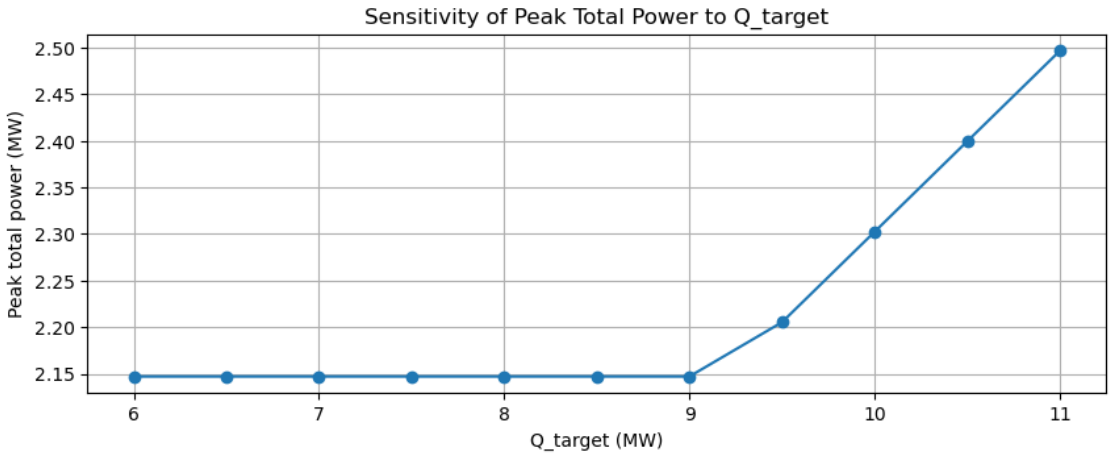

Figure 34

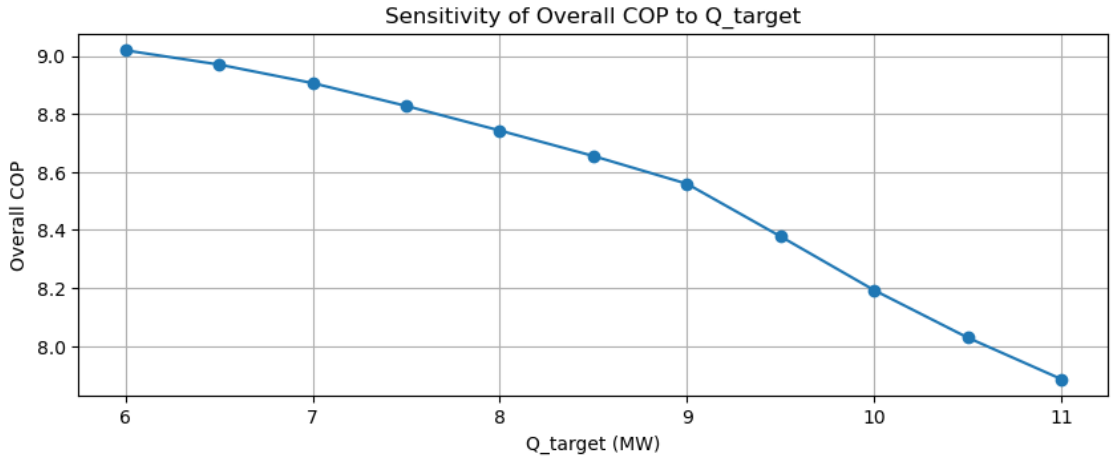

Figure 35

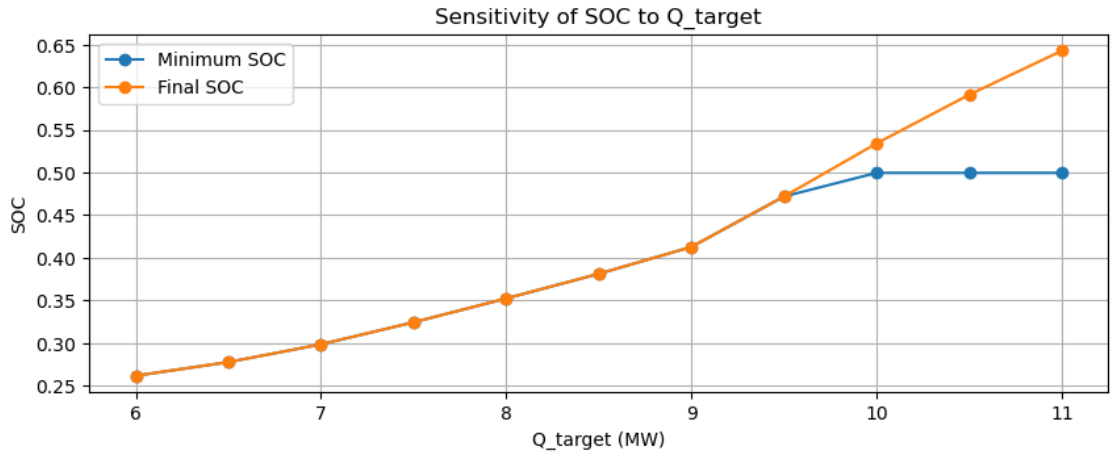

Figure 36

As per figures 37 and 38, $\dot{Q}_{\text{target}}$ should not exceed 9 MW. Figure 39 also tells us that the COP drops past 9 MW and figure 40 tells us that the TES usage drops.

After our sweeps, we now know what is worth optimizing, understand the physics of our system better, and have some bounds to work with. To summarize,

For $T_{\text{discharge}}$, we are deciding the temperature at which we decide to start fighting the peak. We will set our bound as anything between 17°C and 19°C to ensure that the peak load does not get too high.

For T_{charge}, we are deciding how efficiently to prepare our system. We don't want to go too low and waste TES, but we don't want to go too high and miss the peak either. We will thus go in the middle between 10°C and 12°C.

$\dot{Q}_{\text{TES,max}}$, or power capacity, essentially tells us how fast we can shift thermal loads. A value too

low means that the peak remains too high, but there are diminishing returns on peak shaving and COP past a certain point. We will set our bound as between 2.5 MW and 3.0 MW.

Finally, $\dot{Q}_{\text{target}}$ is all about how aggressively we are using TES. At values too high, TES is getting used less because it kicks in past a higher threshold for the chiller load, and the peak shaving is worse. Because the state of charge is too low at low values as well, we will settle for somewhere in the middle, between 8.0 MW and 9.0 MW.

1.4.1 Implementing the Genetic Algorithm

We can now implement the genetic algorithm. We have a general sense of the types of values for our parameters, but we need to find the best combination of them.

We begin with the parameter set

$$\mathbf{x} = \left(T_{\text{discharge}},\ T_{\text{charge}},\ \dot{Q}_{\text{TES,max}},\ \dot{Q}_{\text{target}}\right).$$

The search space is restricted based on insights from the sweeps:

$$T_{\text{discharge}} \in [17, 19]^{\circ}\text{C}$$

$$T_{\text{charge}} \in [10, 12]^{\circ}\text{C}$$

$$\dot{Q}_{\text{TES,max}} \in [2.5, 3.0]\ \text{MW}$$

$$\dot{Q}_{\text{target}} \in [8.0, 9.0]\ \text{MW}$$

The objective is to minimize peak demand while maintaining a high COP if possible and avoiding excessive depletion of TES. This is encoded in a scalar objective function

$$J = w_1 \cdot P_{\text{peak}} + w_2 \cdot \dot{Q}_{\text{chiller,peak}} - w_3 \cdot \text{COP}$$

with additional penalty terms applied if SOC_{min} falls below a safety threshold or the final SOC indicates excessive depletion.

The genetic algorithm proceeds as follows:

1. **Initialization**
   A population of candidate solutions is randomly generated within the parameter bounds.
2. **Evaluation**
   Each candidate is simulated using the TES model, and its objective value is computed.
3. **Selection**
   High-performing candidates are selected based on their objective values.
4. **Crossover**
   New candidate solutions are generated by combining parameters from selected parents.

5. **Mutation**
   Small random perturbations are introduced to maintain diversity and avoid local minima.

6. **Iteration**
   Steps 2–5 are repeated over multiple generations, progressively improving the population.

As the algorithm evolves, the population converges toward a set of parameters that balance what we desire.

[14]:
```
# ----------------------------
# Genetic Algorithm Optimization
# ----------------------------

# Bounds from parameter sweeps
bounds = {
    "T_discharge": (17.0, 19.0),    # deg C
    "T_charge": (10.0, 12.0),       # deg C
    "Q_TES_max": (2.5e6, 3.0e6),    # W
    "Q_target": (8.0e6, 9.0e6)      # W
}

param_names = list(bounds.keys())
lower_bounds = np.array([bounds[p][0] for p in param_names])
upper_bounds = np.array([bounds[p][1] for p in param_names])

# Fixed TES energy capacity
E_max_fixed = 2.0e11

# GA settings
population_size = 40
num_generations = 60
mutation_rate = 0.15
mutation_scale = 0.08
elite_count = 4

np.random.seed(1)

def objective(x):
    T_discharge, T_charge, Q_TES_max, Q_target = x

    # Enforce physically meaningful order
    if T_charge >= T_discharge:
        return 1e9

    peak_chiller, peak_power, overall_COP, min_SOC, final_SOC = run_model_2(
        E_max=E_max_fixed,
        Q_TES_max=Q_TES_max,
        Q_target=Q_target,
        T_charge=T_charge,
```

```
        T_discharge=T_discharge
    )

    # Normalize terms
    peak_power_MW = peak_power / 1e6
    peak_chiller_MW = peak_chiller / 1e6

    # Main objective:
    # lower peak power, lower peak chiller load, higher COP
    score = (
        1.0 * peak_power_MW
        + 0.15 * peak_chiller_MW
        - 0.10 * overall_COP
    )

    # SOC safety penalty
    if min_SOC < 0.30:
        score += 100.0 * (0.30 - min_SOC)**2

    # Avoid ending with tank too depleted
    if final_SOC < 0.40:
        score += 50.0 * (0.40 - final_SOC)**2

    return score


# Initialize population
population = lower_bounds + np.random.rand(population_size, len(param_names)) *␣
 ↪(upper_bounds - lower_bounds)

best_history = []

for gen in range(num_generations):

    scores = np.array([objective(ind) for ind in population])

    # Sort by score
    sorted_idx = np.argsort(scores)
    population = population[sorted_idx]
    scores = scores[sorted_idx]

    best_history.append(scores[0])

    # Elites survive
    new_population = [population[i].copy() for i in range(elite_count)]

    # Generate rest of population
```

```
    while len(new_population) < population_size:

        # Tournament selection
        candidates = np.random.choice(population_size // 2, size=2,␣
 ↪replace=False)
        parent1 = population[candidates[0]]
        parent2 = population[candidates[1]]

        # Crossover
        alpha = np.random.rand(len(param_names))
        child = alpha * parent1 + (1 - alpha) * parent2

        # Mutation
        if np.random.rand() < mutation_rate:
            mutation = np.random.normal(0, mutation_scale,␣
 ↪size=len(param_names))
            child = child + mutation * (upper_bounds - lower_bounds)

        # Clip to bounds
        child = np.clip(child, lower_bounds, upper_bounds)

        new_population.append(child)

    population = np.array(new_population)

# Final best result
scores = np.array([objective(ind) for ind in population])
best_idx = np.argmin(scores)
best_x = population[best_idx]

T_discharge_opt, T_charge_opt, Q_TES_max_opt, Q_target_opt = best_x

peak_chiller_opt, peak_power_opt, overall_COP_opt, min_SOC_opt, final_SOC_opt =␣
 ↪run_model_2(
    E_max=E_max_fixed,
    Q_TES_max=Q_TES_max_opt,
    Q_target=Q_target_opt,
    T_charge=T_charge_opt,
    T_discharge=T_discharge_opt
)

print("========== GA OPTIMIZATION RESULT ==========")
print("Optimal T_discharge:", T_discharge_opt, "deg C")
print("Optimal T_charge:", T_charge_opt, "deg C")
print("Optimal Q_TES_max:", Q_TES_max_opt / 1e6, "MW")
print("Optimal Q_target:", Q_target_opt / 1e6, "MW")
print()
```

```
print("Peak chiller load:", peak_chiller_opt / 1e6, "MW")
print("Peak total power:", peak_power_opt / 1e6, "MW")
print("Overall COP:", overall_COP_opt)
print("Minimum SOC:", min_SOC_opt)
print("Final SOC:", final_SOC_opt)

# Plot convergence
plt.figure(figsize=(10,4))
plt.plot(best_history, marker='o')
plt.xlabel("Generation")
plt.ylabel("Best objective value")
plt.title("GA Convergence History")
plt.grid(True)
add_caption(fig_counter); fig_counter += 1
plt.show()
```

```
========== GA OPTIMIZATION RESULT ==========
Optimal T_discharge: 17.70417144188271 deg C
Optimal T_charge: 11.532980906873869 deg C
Optimal Q_TES_max: 3.0 MW
Optimal Q_target: 8.0 MW

Peak chiller load: 9.0 MW
Peak total power: 2.1081900052975455 MW
Overall COP: 8.689731638332308
Minimum SOC: 0.4019496778304773
Final SOC: 0.4019496778304773
```

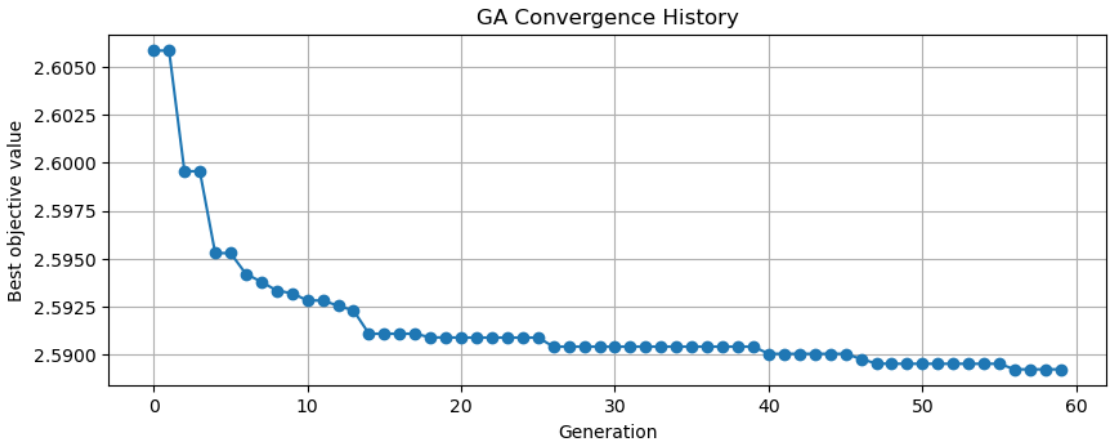

Figure 37

Let's rerun our model with these new values and plot.

[15]:
```
# ----------------------------
# TES Model 2 rerun with GA-optimized values
# ----------------------------

dt = 0.1
dt_sec = dt * 3600.0
t_array = np.arange(0, 24 + dt, dt)

T_bar = 16.0
A_wb = 6.0
phi_wb = 10.0

# Fixed / optimized parameters
E_max_opt = 2.0e11
E_TES = 0.5 * E_max_opt

Q_TES_max_opt = Q_TES_max_opt
Q_target_opt = Q_target_opt
T_charge_opt = T_charge_opt
T_discharge_opt = T_discharge_opt

loss_rate = 0.01 / 24

# Storage arrays for optimized model
T_wb_hist_opt = []
Q_IT_hist_opt = []
Q_econ_hist_opt = []
Q_chiller_hist_opt = []
Q_chiller_eff_hist_opt = []
Q_TES_hist_opt = []
E_TES_hist_opt = []
W_total_base_hist_opt = []
W_total_hist_opt = []
COP_base_hist_opt = []
COP_TES_hist_opt = []
loss_energy_total_opt = 0.0
Q_loss_hist_opt = []

for t in t_array:

    T_wb = T_bar + A_wb * np.sin(2 * np.pi * (t - phi_wb) / 24)
    Q_IT_t = 10e6 + 2e6 * np.cos(2 * np.pi * (t - 16) / 24)

    Q_econ, Q_chiller_base, W_total_base = system_model(T_wb, Q_IT_t)

    # TES standing loss
    E_loss = E_TES * loss_rate * dt
```

```
    E_TES = E_TES - E_loss
    loss_energy_total_opt += E_loss
    Q_loss_hist_opt.append(E_loss / dt_sec)

    # Optimized hybrid TES control
    if T_wb <= T_charge_opt:
        Q_TES_req = -Q_TES_max_opt

    elif T_wb >= T_discharge_opt:
        Q_TES_req = max(0.0, Q_chiller_base - Q_target_opt)

    else:
        Q_TES_req = 0.0

    Q_TES = np.clip(Q_TES_req, -Q_TES_max_opt, Q_TES_max_opt)

    if Q_TES > 0:
        Q_TES = min(Q_TES, E_TES / dt_sec)
    elif Q_TES < 0:
        Q_TES = -min(abs(Q_TES), (E_max_opt - E_TES) / dt_sec)

    E_TES = E_TES + (-Q_TES) * dt_sec
    E_TES = max(0.0, min(E_TES, E_max_opt))

    Q_chiller_eff = max(Q_IT_t - Q_econ - Q_TES, 0.0)

    T_cw_out, T_cond = tower_temperature(T_wb)
    W_comp_eff, Q_rej_eff = chiller(Q_chiller_eff, T_cond)

    _, W_rack = rack(Q_IT_t)
    Q_coil_t, W_crah = crah(Q_IT_t)
    _, W_pumps = chilled_water(Q_coil_t, Q_IT_t)

    W_total_eff = W_rack + W_crah + W_pumps + W_comp_eff

    T_wb_hist_opt.append(T_wb)
    Q_IT_hist_opt.append(Q_IT_t)
    Q_econ_hist_opt.append(Q_econ)
    Q_chiller_hist_opt.append(Q_chiller_base)
    Q_chiller_eff_hist_opt.append(Q_chiller_eff)
    Q_TES_hist_opt.append(Q_TES)
    E_TES_hist_opt.append(E_TES)
    W_total_base_hist_opt.append(W_total_base)
    W_total_hist_opt.append(W_total_eff)
    COP_base_hist_opt.append(Q_IT_t / W_total_base)
    COP_TES_hist_opt.append(Q_IT_t / W_total_eff)
```

```
# Convert to arrays
T_wb_hist_opt = np.array(T_wb_hist_opt)
Q_IT_hist_opt = np.array(Q_IT_hist_opt)
Q_econ_hist_opt = np.array(Q_econ_hist_opt)
Q_chiller_hist_opt = np.array(Q_chiller_hist_opt)
Q_chiller_eff_hist_opt = np.array(Q_chiller_eff_hist_opt)
Q_TES_hist_opt = np.array(Q_TES_hist_opt)
E_TES_hist_opt = np.array(E_TES_hist_opt)
W_total_base_hist_opt = np.array(W_total_base_hist_opt)
W_total_hist_opt = np.array(W_total_hist_opt)
COP_base_hist_opt = np.array(COP_base_hist_opt)
COP_TES_hist_opt = np.array(COP_TES_hist_opt)
Q_loss_hist_opt = np.array(Q_loss_hist_opt)

SOC_hist_opt = E_TES_hist_opt / E_max_opt
```

[16]:
```
plot_tes_results(
    t_array,
    T_wb_hist_opt,
    Q_IT_hist_opt,
    SOC_hist_opt,
    Q_TES_hist_opt,
    Q_chiller_hist_opt,
    Q_chiller_eff_hist_opt,
    W_total_base_hist_opt,
    W_total_hist_opt,
    COP_base_hist_opt,
    COP_TES_hist_opt,
    model_name="Optimized TES Model 2",
    fig_start=38
)
```

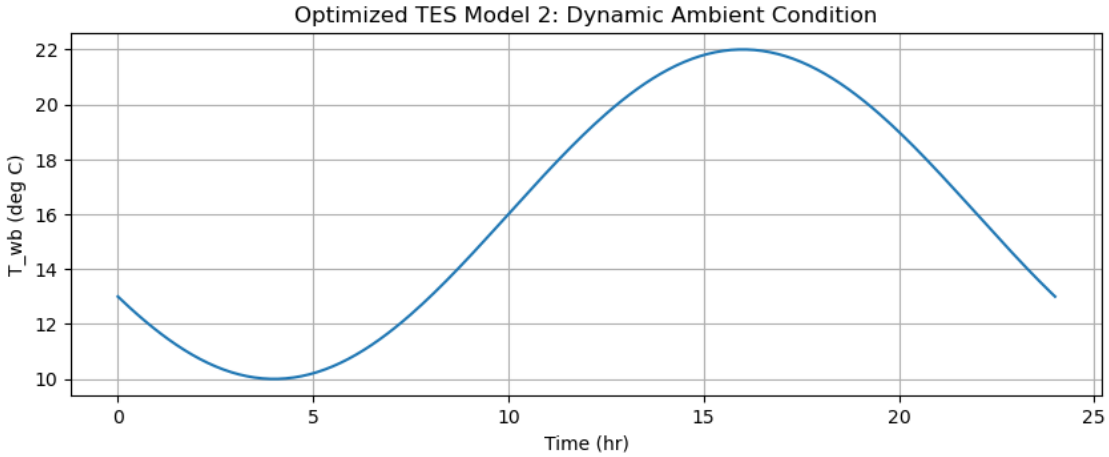

Figure 38

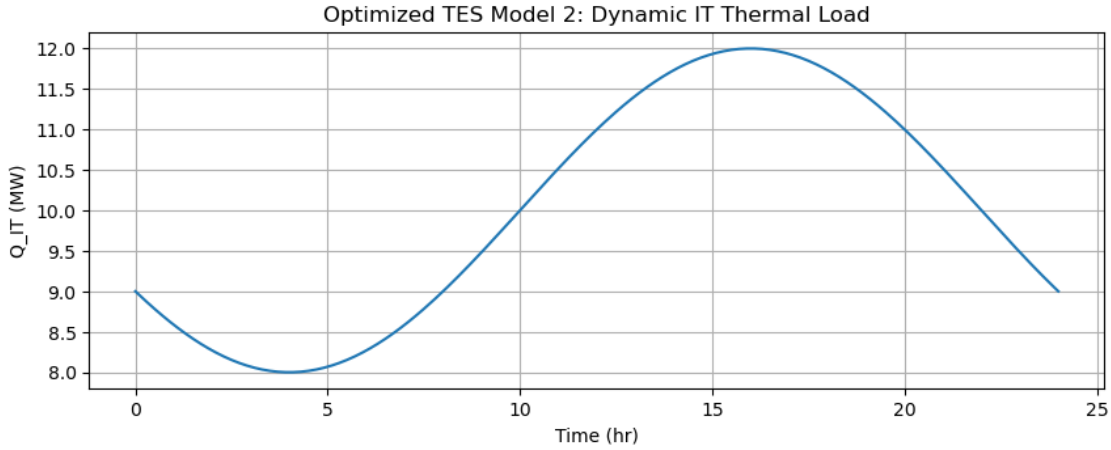

Figure 39

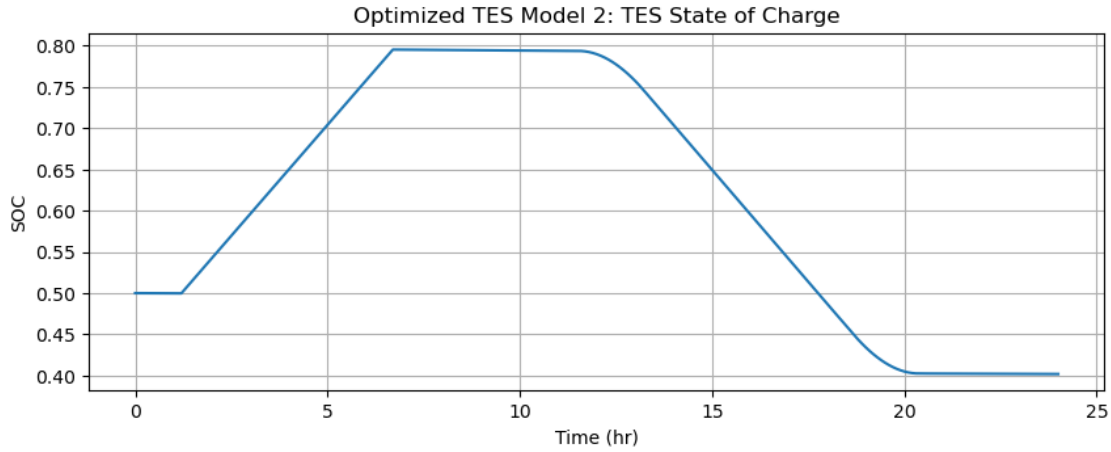

Figure 40

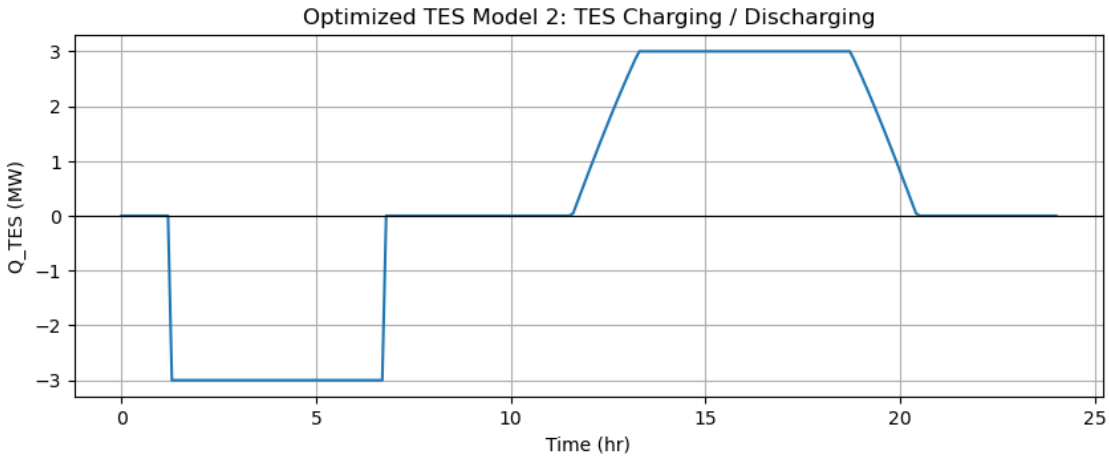

Figure 41

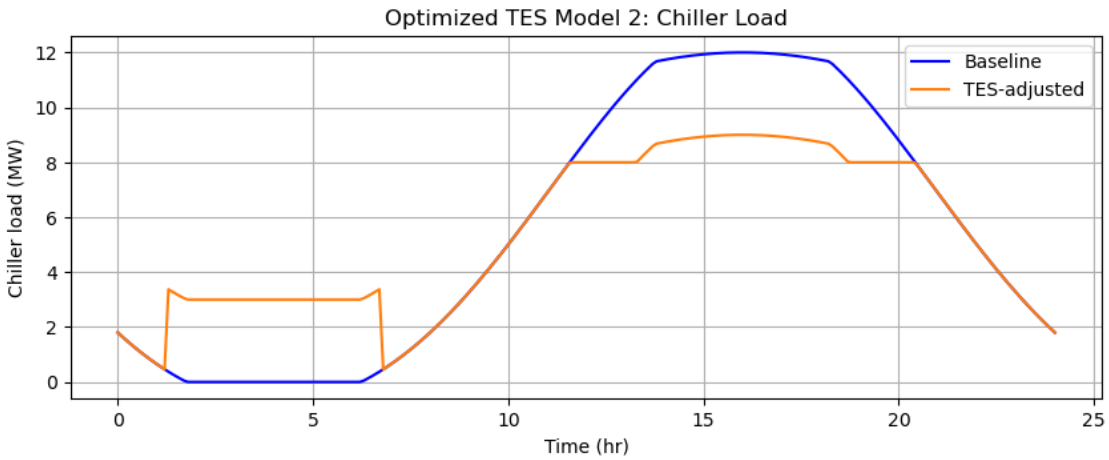

Figure 42

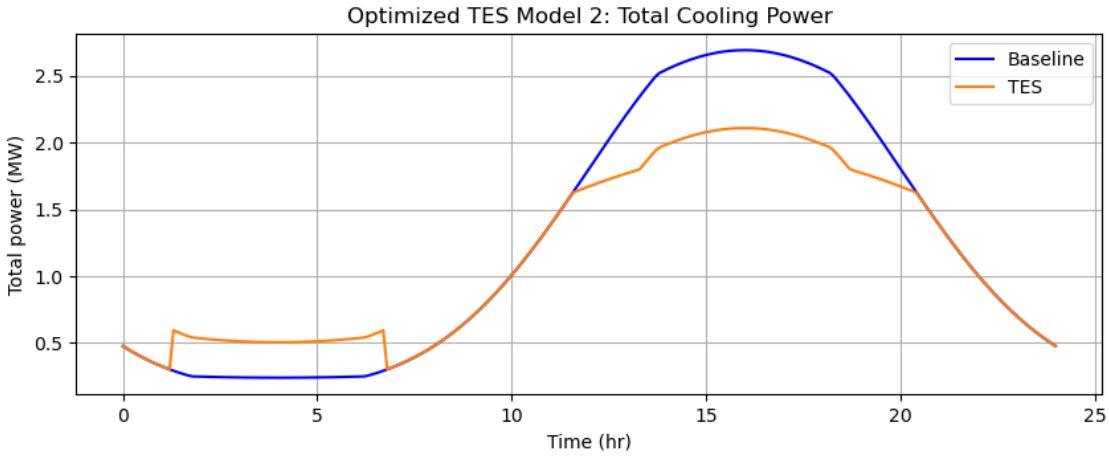

Figure 43

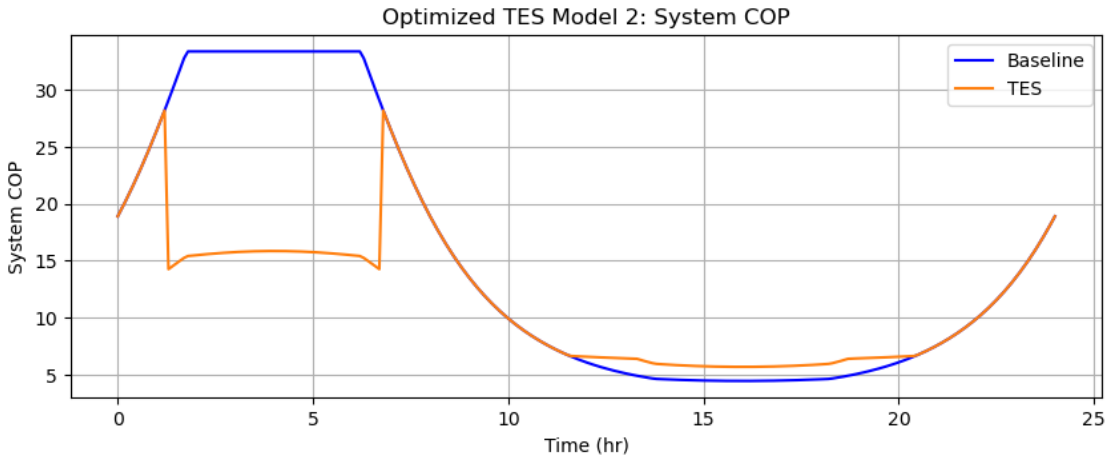

Figure 44

[21]:
```
# ----------------------------
# Compare original Model 2 to optimized Model 2
# ----------------------------

fig_counter = 45

plt.figure(figsize=(10, 4))
plt.plot(t_array, SOC_hist_2, color="green", label="Original TES")
plt.plot(t_array, SOC_hist_opt, color="tab:orange", label="Optimized TES")
plt.xlabel("Time (hr)")
plt.ylabel("SOC")
plt.title("State of Charge Comparison")
```

```
plt.grid(True)
plt.legend()
add_caption(fig_counter); fig_counter += 1
plt.show()

plt.figure(figsize=(10, 4))
plt.plot(t_array, Q_TES_hist_2 / 1e6, color="green", label="Original TES")
plt.plot(t_array, Q_TES_hist_opt / 1e6, color="tab:orange", label="Optimized␣
 ↪TES")
plt.axhline(0, color="black", linewidth=0.8)
plt.xlabel("Time (hr)")
plt.ylabel("Q_TES (MW)")
plt.title("TES Charging / Discharging Comparison")
plt.grid(True)
plt.legend()
add_caption(fig_counter); fig_counter += 1
plt.show()

plt.figure(figsize=(10, 4))
plt.plot(t_array, Q_chiller_hist_2 / 1e6, color="blue", label="Baseline")
plt.plot(t_array, Q_chiller_eff_hist_2 / 1e6, color="green", label="Original␣
 ↪TES")
plt.plot(t_array, Q_chiller_eff_hist_opt / 1e6, color="tab:orange",␣
 ↪label="Optimized TES")
plt.xlabel("Time (hr)")
plt.ylabel("Chiller load (MW)")
plt.title("Chiller Load Comparison")
plt.grid(True)
plt.legend()
add_caption(fig_counter); fig_counter += 1
plt.show()

plt.figure(figsize=(10, 4))
plt.plot(t_array, W_total_base_hist_2 / 1e6, color="blue", label="Baseline")
plt.plot(t_array, W_total_hist_2 / 1e6, color="green", label="Original TES")
plt.plot(t_array, W_total_hist_opt / 1e6, color="tab:orange", label="Optimized␣
 ↪TES")
plt.xlabel("Time (hr)")
plt.ylabel("Total power (MW)")
plt.title("Total Cooling Power Comparison")
plt.grid(True)
plt.legend()
add_caption(fig_counter); fig_counter += 1
plt.show()

plt.figure(figsize=(10, 4))
plt.plot(t_array, COP_base_hist_2, color="blue", label="Baseline")
```

```
plt.plot(t_array, COP_TES_hist_2, color="green", label="Original TES")
plt.plot(t_array, COP_TES_hist_opt, color="tab:orange", label="Optimized TES")
plt.xlabel("Time (hr)")
plt.ylabel("System COP")
plt.title("System COP Comparison")
plt.grid(True)
plt.legend()
add_caption(fig_counter); fig_counter += 1
plt.show()
```

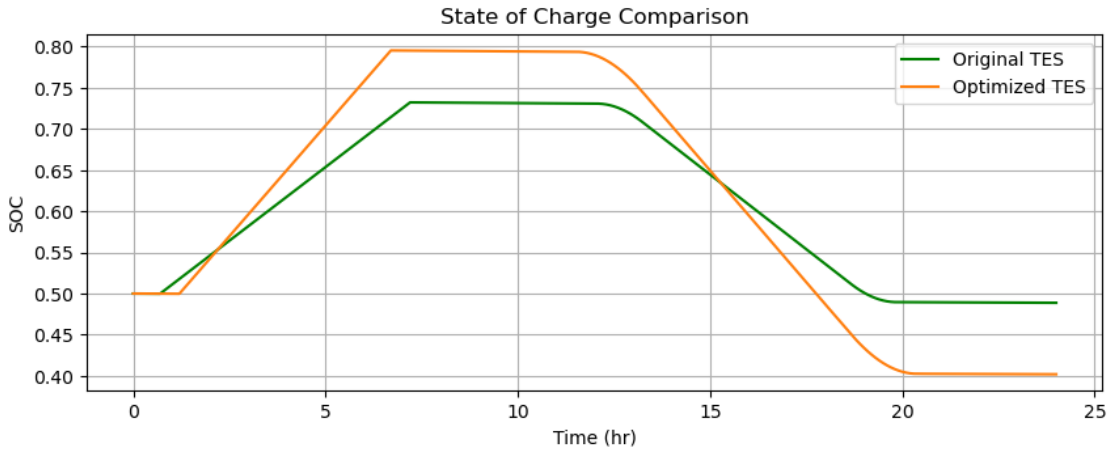

Figure 45

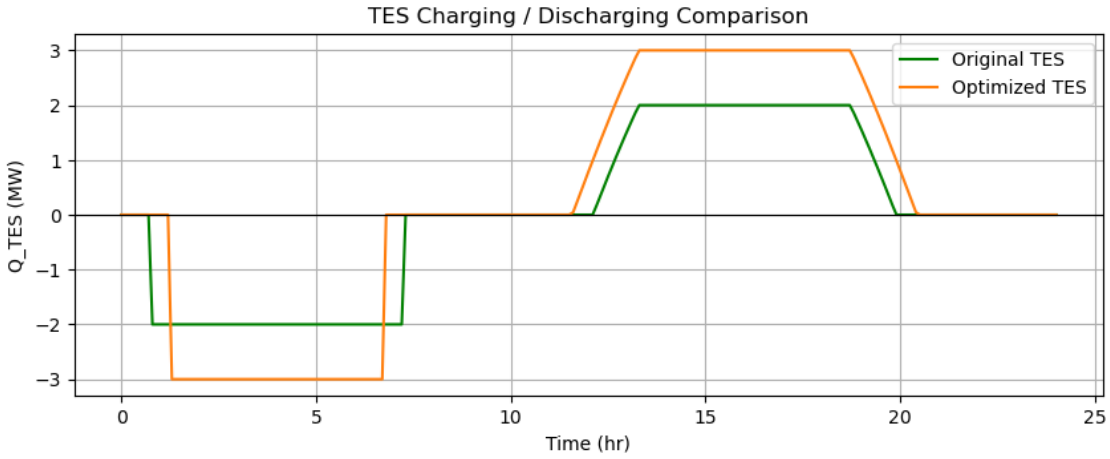

Figure 46

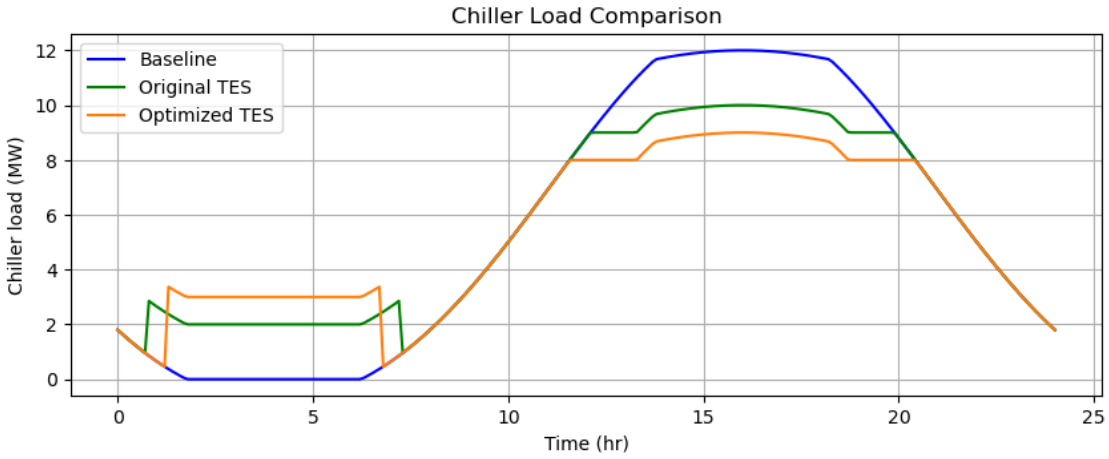

Figure 47

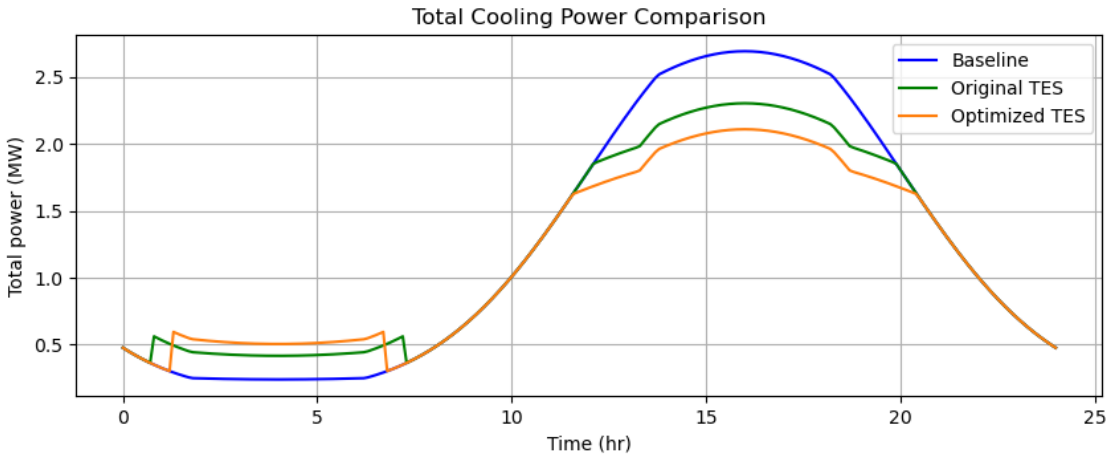

Figure 48

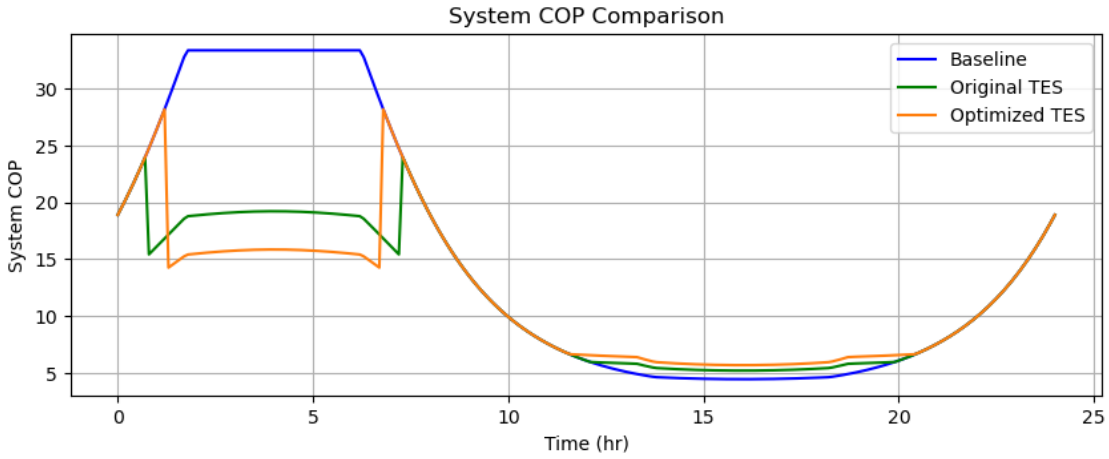

Figure 49

1.5 Testing Our Model with an Unpredictable IT Load

Previously, our model used a dynamic IT load that was based on a peak at 4 pm and thus was very smooth and predictable. In reality, IT loads change rapidly and can experience spikes due to AI demand. To better test our model, we replaced the smooth dynamic load with a stochastic load model:

$$\dot{Q}_{\text{IT}}(t) = \dot{Q}_{\text{smooth}}(t) + \dot{Q}_{\text{background}}(t) + \dot{Q}_{\text{noise}}(t) + \sum_i \dot{Q}_{\text{spike},i}(t)$$

where

- $\dot{Q}_{\text{smooth}}(t)$ represents the baseline cycle,
- $\dot{Q}_{\text{background}}(t)$ represents oscillatory variation,
- $\dot{Q}_{\text{noise}}(t)$ introduces random fluctuations,
- and $\sum_i \dot{Q}_{\text{spike},i}(t)$ represents spikes

The baseline demand still produces a peak at around 4 pm:

$$\dot{Q}_{\text{smooth}}(t) = 10 \times 10^6 + 2 \times 10^6 \cos\left(\frac{2\pi(t-16)}{24}\right)$$

We have random oscillatory background variations to prevent an unrealistically smooth load:

$$\dot{Q}_{\text{background}}(t) = 0.6 \times 10^6 \sin\left(\frac{2\pi(t-9)}{24}\right) + 0.4 \times 10^6 \cos\left(\frac{2\pi(t-13)}{24}\right)$$

Random fluctuations were modeled using Gaussian noise:

$$\dot{Q}_{\text{noise}}(t) = 0.30 \times 10^6 \xi(t)$$

where $\xi(t)$ is a randomly generated dimensionless fluctuation centered around zero. In the code, this is implemented as a seeded random value with mean zero and standard deviation one, scaled by 0.30×10^6 W. This adds short-timescale variability to the IT load without changing the overall daily trends.

Finally, three workloads were modeled as Gaussian spikes to imitate bursts in AI compute demand:

$$\dot{Q}_{\text{spike},i}(t) = A_i \exp\left(-\frac{(t-t_i)^2}{2\sigma_i^2}\right)$$

where

- A_i is the amplitude,
- t_i is the time of day the spike happens,
- and σ_i controls the duration of the spike.

We have a smaller spike in the morning, the largest spike in the afternoon, and a flatter spike in the evening.

[22]:
```
# ----------------------------
# TES Model 2 Optimized and with Dynamic IT Load
# ----------------------------

np.random.seed(1)
dt = 0.1
dt_sec = dt * 3600.0
t_array = np.arange(0, 24 + dt, dt)

T_bar = 16.0
A_wb = 6.0
phi_wb = 10.0

# Optimized parameters
E_max_opt = 2.0e11
E_TES = 0.5 * E_max_opt

Q_TES_max_opt = Q_TES_max_opt
Q_target_opt = Q_target_opt
T_charge_opt = T_charge_opt
T_discharge_opt = T_discharge_opt

loss_rate = 0.01 / 24

# Storage arrays for optimized model under spike IT load
T_wb_hist_spike = []
Q_IT_hist_spike = []
Q_econ_hist_spike = []
Q_chiller_hist_spike = []
Q_chiller_eff_hist_spike = []
```

```
Q_TES_hist_spike = []
E_TES_hist_spike = []
W_total_base_hist_spike = []
W_total_hist_spike = []
COP_base_hist_spike = []
COP_TES_hist_spike = []
loss_energy_total_spike = 0.0
Q_loss_hist_spike = []

for t in t_array:

    T_wb = T_bar + A_wb * np.sin(2 * np.pi * (t - phi_wb) / 24)

    # ------------------------------
    # Variable IT load: stochastic demand + AI workload events
    # ------------------------------

    # Smooth baseline load
    Q_IT_smooth = 10e6 + 2e6 * np.cos(2 * np.pi * (t - 16) / 24)

    # Background workload variation
    background_variation = (
        0.6e6 * np.sin(2 * np.pi * (t - 9) / 8)
        + 0.4e6 * np.sin(2 * np.pi * (t - 13) / 5)
    )

    # Random fluctuations
    noise = np.random.normal(0.0, 0.30e6)

    # Morning
    spike_time_1 = 8.0
    spike_amp_1 = 0.9e6
    spike_width_1 = 0.5
    Q_spike_1 = spike_amp_1 * np.exp(-((t - spike_time_1)**2) / (2 *␣
 ↪spike_width_1**2))

    # Afternoon
    spike_time_2 = 15.0
    spike_amp_2 = 3.0e6
    spike_width_2 = 0.55
    Q_spike_2 = spike_amp_2 * np.exp(-((t - spike_time_2)**2) / (2 *␣
 ↪spike_width_2**2))

    # Evening
    spike_time_3 = 18.0
    spike_amp_3 = 2.0e6
    spike_width_3 = 1.5
```

```
    Q_spike_3 = spike_amp_3 * np.exp(-((t - spike_time_3)**2) / (2 *␣
↪spike_width_3**2))

    # Final IT load
    Q_IT_t = (
        Q_IT_smooth
        + background_variation
        + noise
        + Q_spike_1
        + Q_spike_2
        + Q_spike_3
    )

    Q_IT_t = max(Q_IT_t, 0.0)

    Q_econ, Q_chiller_base, W_total_base = system_model(T_wb, Q_IT_t)

    # TES standing loss
    E_loss = E_TES * loss_rate * dt
    E_TES = E_TES - E_loss
    loss_energy_total_spike += E_loss
    Q_loss_hist_spike.append(E_loss / dt_sec)

    # Optimized hybrid TES control
    if T_wb <= T_charge_opt:
        Q_TES_req = -Q_TES_max_opt

    elif T_wb >= T_discharge_opt:
        Q_TES_req = max(0.0, Q_chiller_base - Q_target_opt)

    else:
        Q_TES_req = 0.0

    Q_TES = np.clip(Q_TES_req, -Q_TES_max_opt, Q_TES_max_opt)

    if Q_TES > 0:
        Q_TES = min(Q_TES, E_TES / dt_sec)
    elif Q_TES < 0:
        Q_TES = -min(abs(Q_TES), (E_max_opt - E_TES) / dt_sec)

    E_TES = E_TES + (-Q_TES) * dt_sec
    E_TES = max(0.0, min(E_TES, E_max_opt))

    Q_chiller_eff = max(Q_IT_t - Q_econ - Q_TES, 0.0)

    T_cw_out, T_cond = tower_temperature(T_wb)
    W_comp_eff, Q_rej_eff = chiller(Q_chiller_eff, T_cond)
```

```
    _, W_rack = rack(Q_IT_t)
    Q_coil_t, W_crah = crah(Q_IT_t)
    _, W_pumps = chilled_water(Q_coil_t, Q_IT_t)

    W_total_eff = W_rack + W_crah + W_pumps + W_comp_eff

    T_wb_hist_spike.append(T_wb)
    Q_IT_hist_spike.append(Q_IT_t)
    Q_econ_hist_spike.append(Q_econ)
    Q_chiller_hist_spike.append(Q_chiller_base)
    Q_chiller_eff_hist_spike.append(Q_chiller_eff)
    Q_TES_hist_spike.append(Q_TES)
    E_TES_hist_spike.append(E_TES)
    W_total_base_hist_spike.append(W_total_base)
    W_total_hist_spike.append(W_total_eff)
    COP_base_hist_spike.append(Q_IT_t / W_total_base)
    COP_TES_hist_spike.append(Q_IT_t / W_total_eff)

T_wb_hist_spike = np.array(T_wb_hist_spike)
Q_IT_hist_spike = np.array(Q_IT_hist_spike)
Q_econ_hist_spike = np.array(Q_econ_hist_spike)
Q_chiller_hist_spike = np.array(Q_chiller_hist_spike)
Q_chiller_eff_hist_spike = np.array(Q_chiller_eff_hist_spike)
Q_TES_hist_spike = np.array(Q_TES_hist_spike)
E_TES_hist_spike = np.array(E_TES_hist_spike)
W_total_base_hist_spike = np.array(W_total_base_hist_spike)
W_total_hist_spike = np.array(W_total_hist_spike)
COP_base_hist_spike = np.array(COP_base_hist_spike)
COP_TES_hist_spike = np.array(COP_TES_hist_spike)
Q_loss_hist_spike = np.array(Q_loss_hist_spike)

SOC_hist_spike = E_TES_hist_spike / E_max_opt
```

[23]:
```
plot_tes_results(
    t_array,
    T_wb_hist_spike,
    Q_IT_hist_spike,
    SOC_hist_spike,
    Q_TES_hist_spike,
    Q_chiller_hist_spike,
    Q_chiller_eff_hist_spike,
    W_total_base_hist_spike,
    W_total_hist_spike,
    COP_base_hist_spike,
    COP_TES_hist_spike,
    model_name="TES Model 2 Optimized with Spiked IT Load",
```

```
    fig_start=50
)
```

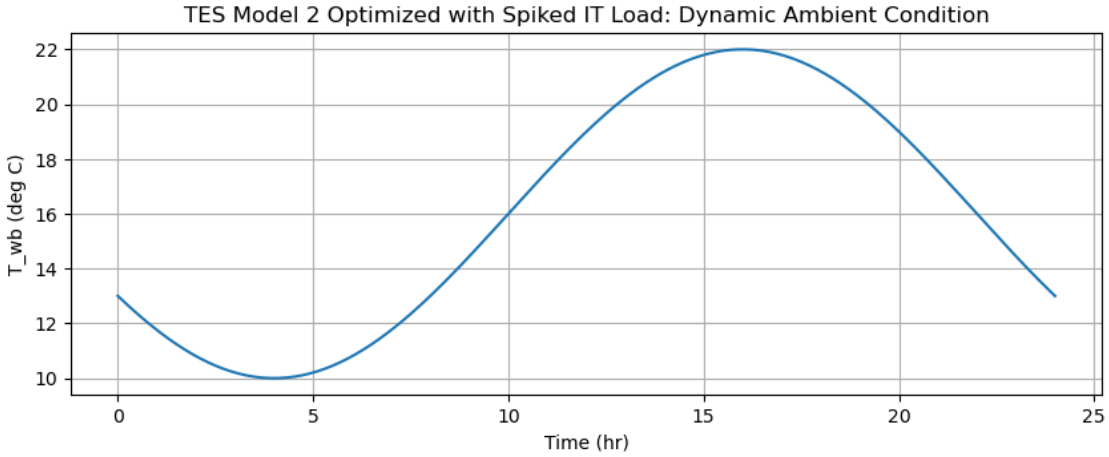

Figure 50

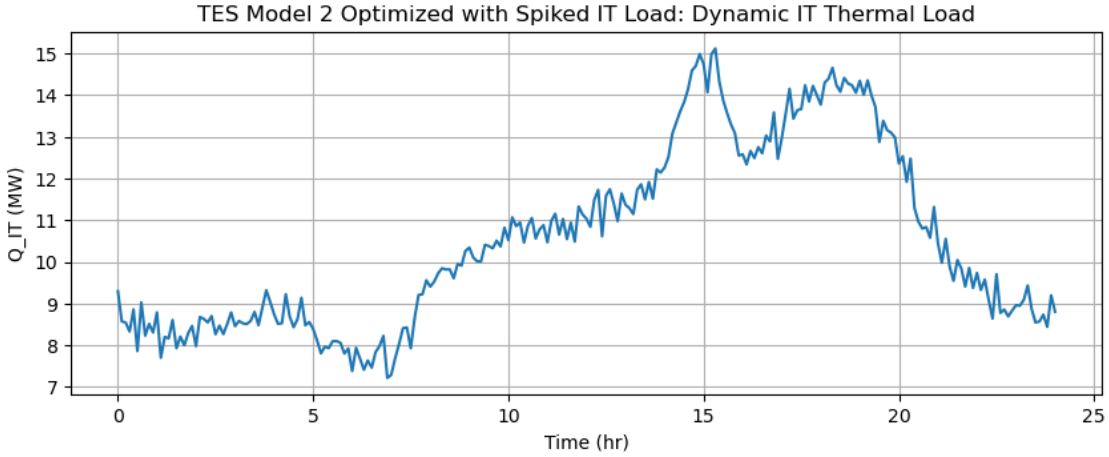

Figure 51

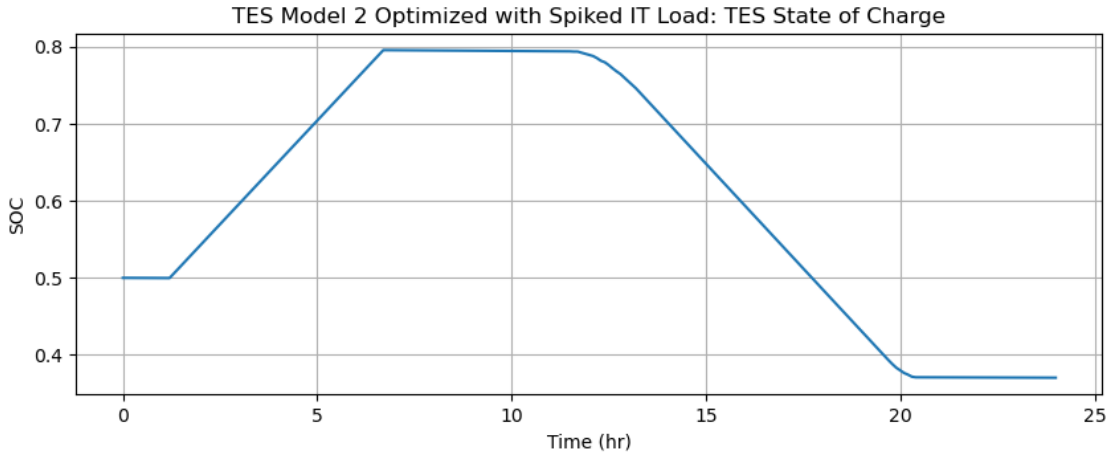

Figure 52

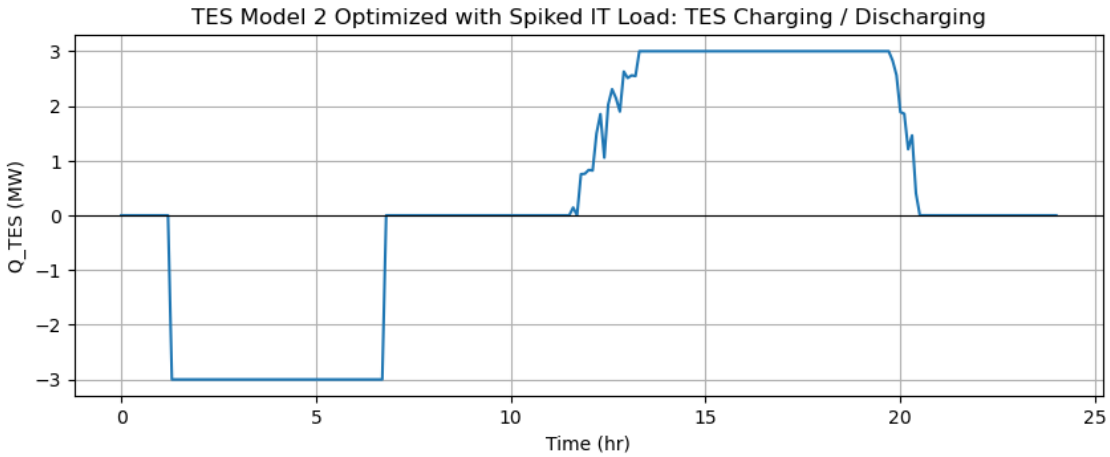

Figure 53

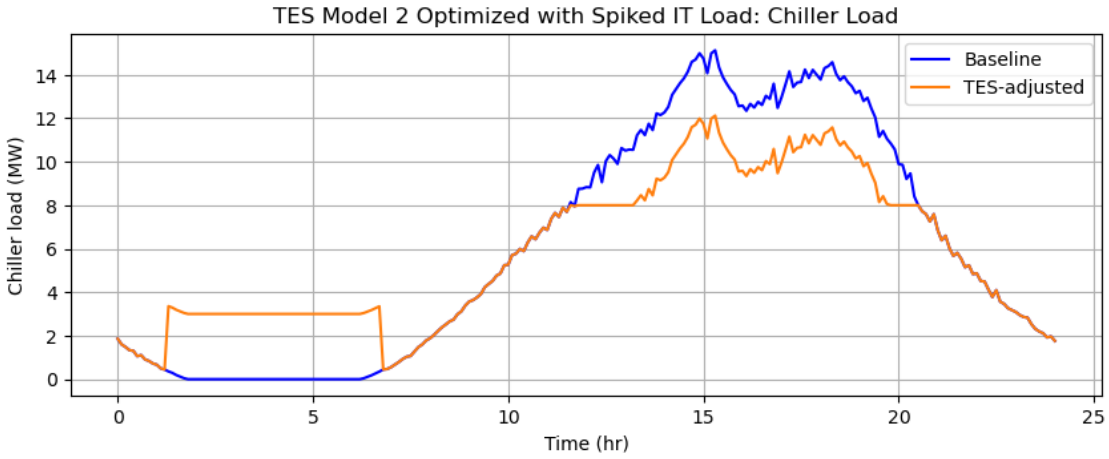

Figure 54

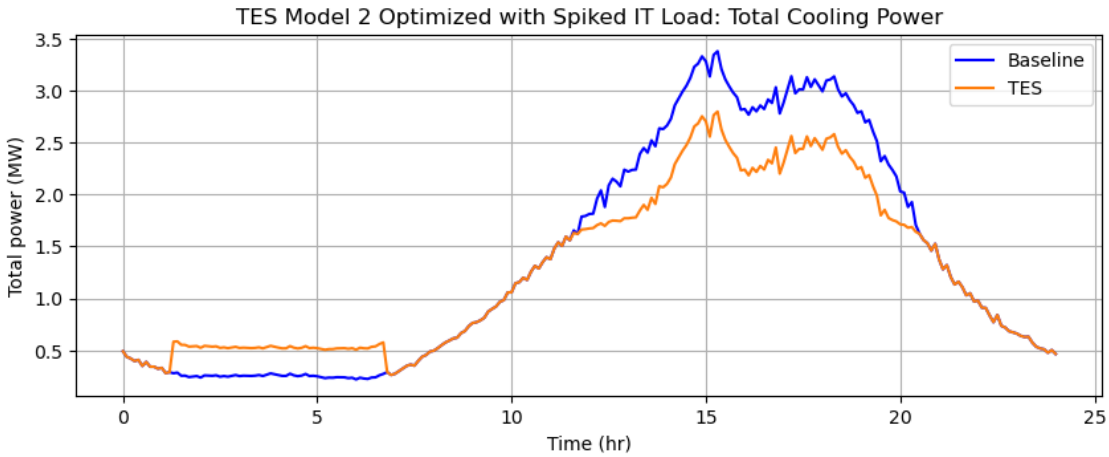

Figure 55

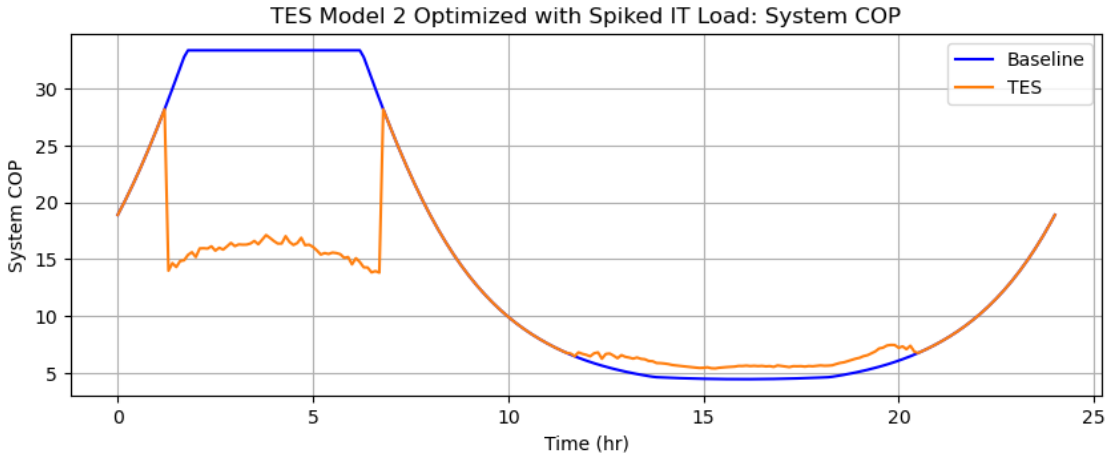

Figure 56

[20]:
```
idx_peak_base = np.argmax(W_total_base_hist_spike)
idx_peak_tes = np.argmax(W_total_hist_spike)
idx_peak_it = np.argmax(Q_IT_hist_spike)

print("Peak IT load:", Q_IT_hist_spike[idx_peak_it] / 1e6, "MW at",␣
 ↪t_array[idx_peak_it], "hr")
print("Baseline peak power:", W_total_base_hist_spike[idx_peak_base] / 1e6, "MW␣
 ↪at", t_array[idx_peak_base], "hr")
print("TES peak power:", W_total_hist_spike[idx_peak_tes] / 1e6, "MW at",␣
 ↪t_array[idx_peak_tes], "hr")
print("Peak baseline chiller:", np.max(Q_chiller_hist_spike) / 1e6, "MW")
print("Peak TES chiller:", np.max(Q_chiller_eff_hist_spike) / 1e6, "MW")
print("Min SOC:", np.min(SOC_hist_spike))
print("Final SOC:", SOC_hist_spike[-1])
```

```
Peak IT load: 15.11978200426622 MW at 15.3 hr
Baseline peak power: 3.37709788543446 MW at 15.3 hr
TES peak power: 2.797029120326291 MW at 15.3 hr
Peak baseline chiller: 15.11978200426622 MW
Peak TES chiller: 12.11978200426622 MW
Min SOC: 0.37054400164062457
Final SOC: 0.37054400164062457
```

While our load is still an approximation, it is still reasonable for a medium-to-large data center. With this more unpredictable load with multiple spikes, we were able to reduce the peak chiller from 15.1 MW to 12.1 MW, which is an impressive and meaningful result for the viablity of thermal energy storage as a method to reduce peak cooling demand.

1.6 Discussion of Further Research

Modeling a hyperscale data center is significantly more complex. Head loss becomes important because of the scale of infrastructure and the workloads are much more difficult to model because of the variability of different compute clusters. It would require us to use a digital twin.

While this project uses cold storage, hot storage has applications in data centers when it comes to serving as an alternative to UPS and batteries as emergency power. Furthermore, there is potential to couple hot storage with a concentrated solar power systems or district heating networks.

Model latent or thermochemical thermal energy storage and compare their performance against sensible heat storage.

Determine whether ocean or orbital data centers are viable or not.

1.7 References

[1] O. Hyunjun, J. Wencheng, et al. Techno-economic performance of reservoir thermal energy storage for data center cooling system, 2025.

[2] D. Garday, J. Housley, Thermal Storage System Provides Emergency Data Center Cooling, 2007.

[3] C. Halloran, Q. Luo, Grid Impact of Reservoir Thermal Energy Storage for Data Center Cooling, 2026.

[4] C.H. Ho, A. Ambrosini, U.S. DOE Energy Storage Handbook, 2026.